www.freesoftwaremagazine.com

Issue 4 - May, 2005

Free Software MAGAZINE

FOCUS: THE RISKS OF PROPRIETARY SOFTWARE

(MATT BARTON)

THE RISKS OF WRITING PROPRIETARY SOFTWARE

(MATT BARTON)

THE RISK OF USING PROPRIETARY SOFTWARE

(MARTIN C. BROWN)

FINDING ALTERNATIVES IN DEVELOPING SOFTWARE

(ADAM HYDE)

net.labels

AN INTRODUCTION TO THE EMERGING PHENOMENON OF NET.LABELS

(DAVID N. WELTON)

Make it right using Tcl

SOFTWARE TESTING WITH TCL FOR APACHE RIVET

(MARCUS MCCALLION)

Graphic icons

SYMBOLS OF AUTHORITY, POWER AND CONTROL

(FREDERICK NORONHA)

The social implications of free software

IN THE NOT-SO-AFFLUENT WORLD IN PARTICULAR, PROPRIETARY SOFTWARE DESERVES TO BE SKIPPED

(JOHN LOCKE)

Worst case scenario - protecting your computer

HOW TO KEEP SENSITIVE INFORMATION SAFE

COVER BY WWW.QREATE.NET

qreate

ISSN 1746-8752

9 771746 875009 04

Yocto-24

OSS Business Solutions

ATTENTION: SENIOR AND IT EXECUTIVES. GAIN FREEDOM AND SAVE THOUSANDS OF DOLLARS!

ELIMINATE PROPRIETARY SOLUTIONS BY IMPLEMENTING AN OPEN SOURCE SOFTWARE STRATEGY!

RECEIVE A **FREE** OPEN SOURCE KNOWLEDGE STARTER PACK CONTAINING EVERYTHING YOU NEED TO KNOW ABOUT OPEN SOURCE SOFTWARE, ITS BENEFITS AND HOW TO MAKE IT WORK IN YOUR BUSINESS.

CONTACT US **BEFORE 1 APRIL 2005** AND ALSO RECEIVE A FREE OPEN SOURCE ASSESSMENT AND A CONSULTATION SESSION **WORTH $500** FROM YOCTO'S BEST OPEN SOURCE SOFTWARE GURU (GAUTENG, RSA ONLY) WITH NO OBLIGATION AT ALL.

YOCTO OSS HAS THE KNOWLEDGE, EXPERIENCE, AND THE SKILLS YOU REQUIRE TO IMPLEMENT SUCH A STRATEGY. YOCTO OSS PARTNERS WITH MAJOR PLAYERS IN OPEN SOURCE SOFTWARE AND IS ACTIVELY INVOLVED IN THE OPEN SOURCE SOFTWARE RESEARCH COMMUNITY.

CONTACT US VIA OUR WEBSITE HTTP://WWW.YOCTO.CO.ZA OR EMAIL MARKETING@YOCTO.CO.ZA WITH YOUR CONTACT DETAILS AND THE REFERENCE NUMBER FSM001 TO RECEIVE YOUR FREE OPEN SOURCE KNOWLEDGE STARTER PACK.

Make use of open source software to fully support your business strategy.

Do it without any risks to your business!

New vision of Java

Want to buy the **Software 2.0** magazine, please visit our shop at **www.shop.software.pl**

Free Software MAGAZINE

Contents

Issue 4, May 2005

EDITORIAL

What is the next (r)evolution? 7

Will free software be able to join in when it happens?

STARTERS

Unix Power Tools 3rd edition *by Shelley Powers* et al. 9

by Gianluca Pignalberi

FOCUS

The risks of writing proprietary software 10

by Matt Barton

Concrete economical reasons for avoiding proprietary software development

The risk of using proprietary software 17

by Matt Barton

Do you know what you're feeding your computer?

Finding alternatives in developing software 21

by Martin C Brown

Comparing free development tools and libraries with proprietary ones

TECH WORLD

A server for education 27

by Georges Khaznadar

Wims is a "magic" server

Worst case scenario - protecting your computer 35

by John Locke

How to keep sensitive information safe

Make it right using Tcl 40

by David N. Welton

Software testing with Tcl for Apache Rivet

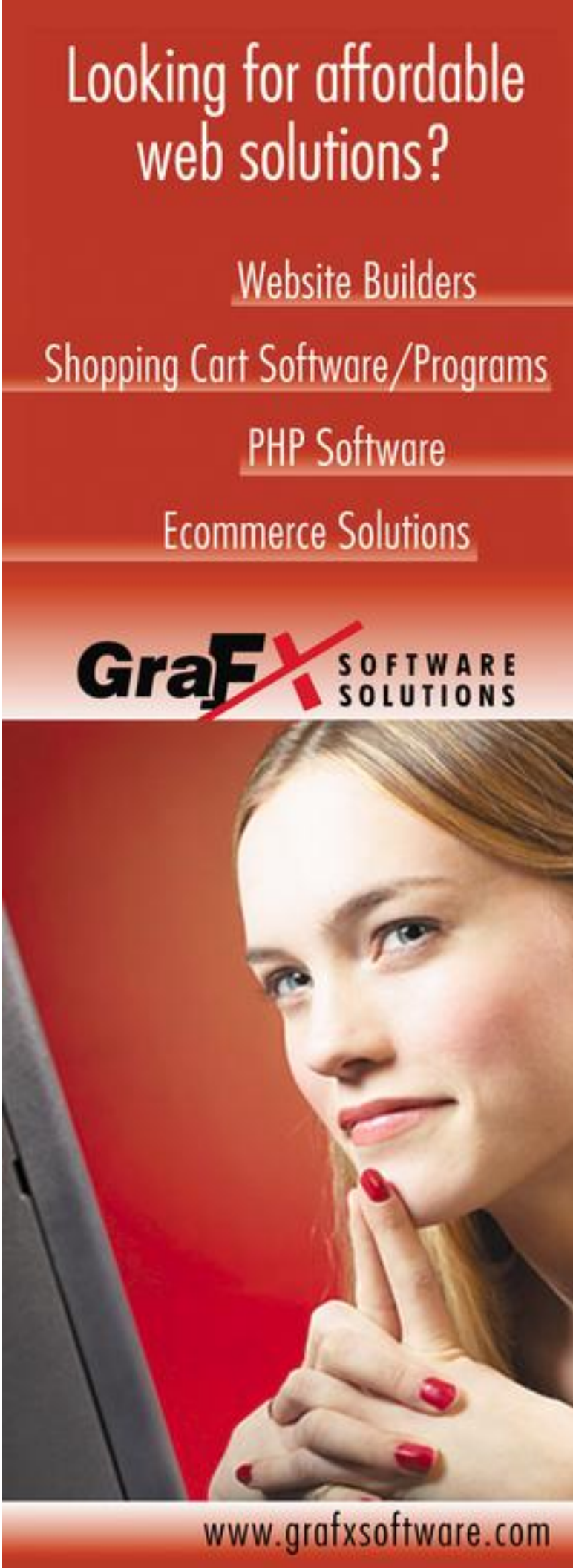
WORD WORLD

net.labels 48
by Adam Hyde
An introduction to the emerging phenomenon of net.labels

Graphic icons 55
by Marcus McCallion
Graphic icons: symbols of authority, power and control

Free software: working together 58
by Albert Witteveen
Avoiding headaches and lawsuits by working together with free software

The social implications of free software 62
by Frederick Noronha
In the not-so-affluent world in particular, proprietary software deserves to be skipped



Are your systems *OPEN* for business?

Our customers' are!

For over 10 years, OCI has been helping organizations build their mission critical infrastructure with open architectures using standards-based OO technologies.

OCI provides the following commercial-grade products to the open-source community:

- ACE - The high performance C++ portability layer combining socket-level performance with type-safe programming, across a wide range of operating systems.
- TAO - An open-source C++ CORBA 2.6 implementation with the widest range of services and span of platforms of any ORB.
- JacORB - The open-source Java CORBA 2.4 implementation.
- MPC - An open-source cross-platform build tool - "script once, build many"
- OVATION - A tool for instrumenting & visualizing CORBA systems behavior.

Can't wait to go open? OCI offers a comprehensive set of migration support services that will help your organization minimize risk, cost, and time when moving to these open-source solutions.

To learn more about these products & services, visit: www.theaceorb.com

Product Development • Consulting • Educational Services

Object Computing, Inc. (OCI)

St. Louis, MO Headquarters: +1 (314) 579-0066
Phoenix, AZ Office: +1 (480) 752-0042

www.ociweb.com

OBJECT COMPUTING, INC.

What is the next (r)evolution?

Will free software be able to join in when it happens?

I am not sure if it's correct to talk about the internet as a revolution. The internet is in fact the result of a slow, hard earned evolution which has lasted about 30 years (!). Slowly, during these years, the costs of laying cables has dropped, the CPU was... well, *invented* (in 1974, the Intel 4004), processing power and memory have increased exponentially and the basic protocols were created (in 1972, the *telnet* protocol).

Maybe, it would be fair to consider the internet a slow, gradual evolution which has caused a sudden, more drastic revolution. The signs that this revolution would take place were all there, and yet I have the feeling that very few people back then would have believed that by the year 2005 the internet would become the most important information infrastructure in the world, with store fronts, credit card fraud, underground peer-to-peer networks, online banking, online repositories for free and non-free software, and so on. But here we are, in the midst of it all, enjoying its benefits and facing newly created problems.

For some reason, I can't stop asking myself: so, what's next? (I can't help it: I imagine a reader of this magazine in about 20 years knowing the answer to my question, but having no way of coming here, back in 2005, and telling me!)

The signs are all here. One thing is becoming faster. Another thing is becoming cheaper. And again something else is becoming more and more advanced. All of this will lead to a sudden, drastic revolution that will change the way we live. But what are these crucial "things"?

I have a few ideas (and this is when my reader in the future starts feeling embarrassed for me, and thinks "please don't, no please don't...).

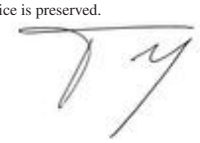
The most important one in my opinion is a revolution in the interface between us and our computers. The way we interact with computers today is unintuitive, cumbersome, even pathetic, and it will need to change drastically in some way. I think keyboards and mice have long passed their due dates; voice interfaces simply don't cut it - and I don't think they *would* cut it even if they were 100% reliable. (How do you create a complex spreadsheet using just your voice?) A few years ago I was a great believer in VR (Virtual Reality: does anybody still remember it?), and I could have sworn that VR would be it, the future - I was wrong when I thought it back then, and I would be wrong if I said it now, at least considering its current incarnations.

I believe that the next revolution could be in the use of neural interfaces, which will allow us to interact with computers using our brains: no "middle man", no meat involved. (Now, it's time for our future reader to burst into laughter and start feeling really sorry for me...). What would we "see"? How would we do anything in there? How would we create a complex spreadsheet just using our brain? Would those "neural chips" be able to make us see, hear, touch, and move? I don't know the answer to these questions. It will largely depend upon what the technology will be able to give us, and how. However, I must admit that I am in fact imagining an advanced version of the infamous VR, which I just said had already failed in the *previous paragraph*...

Finally, there is a more important question which begs to be asked: if we do experience a drastic revolution in the way we use computers (neural interfaces, or whatever else might come along), will free software run the risk of being left out of the picture? If there was a patent which demanded thousands of dollars and a signature on a nasty NDA in order to license the technology and create a "neural application", what would free software developers do?

Would IBM come and rescue us again? Your guess is as good as mine.

Copyright information © 2005 by Tony Mobily
Verbatim copying and distribution of this entire article is permitted in any medium without royalty provided this notice is preserved.



Free Software Magazine is a magazine by **The Open Company Partners Inc**, 90 Main St. Road Town, Tortola BVI

EDITOR IN CHIEF
Tony Mobily (t.mobily@)

TECHNICAL EDITORS
Clare James (c.james@)
Pancrazio De Mauro (p.demauro@)
Gianluca Insolvibile (g.insolvibile@)

EDITORS
Anna Dymitr Hawkes (a.dymitrhawkes@)
Dave Guard (d.guard@)

TECHS
Gianluca Pignalberi (LaTeX class and magazine generation) (g.pignalberi@)
Gian Maria Ricci (RTF to XML converter using VBA) (gm.ricci@)

GRAPHIC DESIGN
Alan Sprecacenere (Web, cover and advertising design) (a.sprecacenere@)
Tony Mobily, Gianluca Pignalberi, Alan Sprecacenere (Magazine design)

THIS PROJECT EXISTS THANKS TO
Donald E. Knuth, Leslie Lamport, People at TeX Users Group TUG (http://www.tug.org)

Every listed person is contactable by email. Please just add freesoftwaremagazine.com to the person's username in parentheses.

For copyright information about the contents of Free Software Magazine, please see the section "Copyright information" at the end of each article. Unless the license is applied immediately with the writing "(The following license is effective immediately)", for six weeks after publication you may not reproduce or retransmit the article, in whole or in part, in any manner, without the prior written consent of the author; six weeks after publication, the license at the end of each article applies.

Open Source Labs

Pioneers in Open Source Solutions

"Open Source Labs, a division of VYOM Labs, is a centre of excellence providing World-Class Support, Consultancy and Training across the entire spectrum of Open Source Technologies. Open Source Labs enables organisations worldwide to gain maximum value from Open Source Software while reducing the total cost of ownership".

Open Source Support Services

We provide migration, installation and post installation support solutions for most Open Source Technologies :

- Linux and MySQL
- Software Engineering tools like CVS, BugZilla
- Software IT management tools like Nagios, MRTG, OTRS
- Messaging solutions including Qmail, Sendmail, Webmail, SpamAssasin, Thunderbird
- Custom Open Source development in PHP, Perl (using MySQL or Postgres SQL servers)

Linux Training

- Linux OS Administration
- Networking and Security
- RedHat Linux
- SuSe Linux

Developer Course on Linux

- OSL 101 Unix Shell scripting
- OSL 102 PHP/Perl + MySQL

Open Source Labs
power of freedom

redhat®
TRAINING PARTNER
Authorized Red Hat Training Partner

vyomlabs
Looking beyond the Horizon

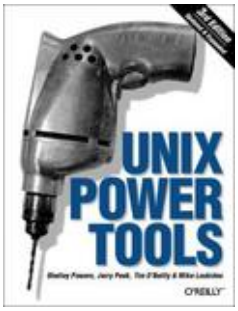
Open Source Labs, (A division of VYOM labs Pvt. Ltd.), Dayaprabha House, Opp Sulzer India, ITI Road, Aundh, Pune 411007, INDIA. Ph: +91 982278700, +91 20 2588 9236
e- mail : contactus@opensource-labs.com

www.opensource-labs.com

Paradigm Plus

Unix Power Tools 3rd edition *by Shelley Powers* et al.

Gianluca Pignalberi



Using a Unix system requires a lot of knowledge, and it's common to see Unix users and administrators spending a lot of time reading handbooks, tutorials and man pages to find out the "right" sequence of keystrokes. In the publishing world there is a little pearl, a single source of information about Unix and how to use it: Unix Power Tools, published by O'Reilly and Associates. O'Reilly is a well known publisher of Unix books; in this one, you'll see Tim O'Reilly himself as an author!

The contents

This book collects a lot of the best practices, advice and rules of thumb for using Unix better, faster, and more effectively. The book's cover states that the authors are Shelley Powers (a new acquisition for this 3rd edition), Jerry Peek, Tim O'Reilly (himself!) and Mike Loukides. However, a work of this size required the effort of more than 40 contributors, all listed within the book. The book is made up of chapters, sections, notes within sections, and each piece has its author specified.

Who's this book for?

This book is for everybody. It's much more than a handbook: it describes techniques and tricks to improve users' proficiency with Unix; it also reveals some of the deeper aspects of Unix, in order to explain why a particular version of Unix is better than another one.

It's much more than a hypertext; one section, for example, explains a command (or several commands) that can be used to perform a job, but also provides a link, which points to a set of related sections elsewhere in the book. So, you can jump through the book following your curiosity or your needs.

You can read this book in three ways: you can start from page one and go through to the end; you can open a random page and read —you'll find something interesting; or you can go to the index and pick a subject. In any case, following the proposed link is often a good idea.

Pros

Suppose that you're working on a Unix workstation and have to solve a problem while using a tool (you name it: AWK, bash, CVS etc.). Which book would you look for? A set of one thousand, specialized books, or a single one-thousand-paged book?

Moreover, in this book you will find information about several Unix flavours (including Linux, BSD, Mac OS X) and a collection of real tools you need to know in order to become a power user (for example: do you know how to down-case a bunch of files in a directory? There is a solution for you). This book is the best way to invest $70.

Cons

Compared to the 2nd edition, this book has some minor deficiencies. First of all, it doesn't have the CD-ROM; if you want to use the source code shown in the book, you'll have to go to the publisher's site and download it. The links are now grey, in the second edition, they were cyan: I found them more difficult to see in this new edition. The cover is now softer than the 2nd edition's, and some of the content has disappeared from the updated sections.

In short

Title	Unix Power Tools 3rd edition
Author	Shelley Powers, Jerry Peek, Tim O'Reilly and Mike Loukides
Publisher	O'Reilly and Associates
ISBN	0596003307
Year	2002
Pages	1151
CD included	No
Mark (out of 10)	9

Copyright information

© 2005 by Gianluca Pignalberi

Verbatim copying and distribution of this entire article is permitted in any medium without royalty provided this notice is preserved.

The risks of writing proprietary software

Concrete economical reasons for avoiding proprietary software development

Matt Barton

Every software developer faces a choice when deciding how to release a new software product. That choice is whether the program will be free or non-free. Unfortunately, many otherwise knowledgeable programmers aren't sure just what this choice means, and may complain that programmers with families really don't have a choice at all — if they want to earn a living, they must charge for their work. However, free software is not about giving software away without cost. Rather, free software is simply an ethical choice that guarantees the freedom of users — and, perhaps more importantly, the freedom of the developer. This last point is often lost, even in discussions among free software developers about the benefits of publicly licensed software.

Nevertheless, a developer who chooses a free software license is not necessarily acting under purely selfless motives - there are concrete economical reasons for doing so. I will first discuss the immediate benefits of releasing free software to developers, then discuss broader advantages to the industry and society.

The private benefits of public licensing

I recently had the opportunity to interview two game developers: Daniel Horn and Mike Boeh. Horn chose to release his popular *Privateer* remake, *VegaStrike,* under the General Public License, whereas Boeh has achieved the "holy grail" of independent game development - making enough money to adequately support his family - by releasing a series of high-quality proprietary action games. Both developers are highly skilled and successful at marketing their work, and though their programs offer the player substantially different experiences, their games are highly polished and professional - and a good deal more fun to play than many big-budget "A-List" titles.

Fig. 1: VegaStrike is one of the most sophisticated GNU games available

Comparing the development strategies of these programmers is quite revealing. Horn had a stunning revelation when he learned about GNU/Linux and the unique way it had been built. Horn, who had intended *VegaStrike* to be a standard proprietary product, decided then to embrace the free software model. "I realized that this was how it had to

be. This is what would differentiate me from my competition," says Horn. This decision proved to be the right one. Soon after he announced his game, another programmer who had been working on another *Privateer* remake heard of it and promptly decided to abandon his own project and dedicate his time and energy to *VegaStrike*. Graphic artists, musicians, and writers soon followed, and now *VegaStrike's* development is largely the responsibility of contributors volunteering their work. This is possible because users have access to *VegaStrike's* source code, or the "human-readable" scripts that tell computers what to do. If Horn had only released the binary code, or the "machine-readable" scripts, the users would have been unable to reliably discern how the program worked or contribute to it. Thus, by *sharing* his source code, Horn has received help from hundreds of users who gladly shared their resources and talents with him.

Mike Boeh's approach to software development differs sharply from Horn's. Most significantly, Boeh's games are released only in binary form. Furthermore, though Boeh offers "playable demos" of his titles, users must purchase the "full versions" and are not allowed to share them. While Boeh prides himself on the originality and versatility of his games, he admitted to me that he does share an engine, or the "core code" of his games, with another proprietary developer. However, Boeh does not choose to share this engine or his code with other developers. Instead, Boeh contracts out for artistic and music talent and either pays them up front or promises them a share of his royalties.

> A developer who chooses a free software license is not necessarily acting under purely selfless motives - there are concrete economical reasons for doing so

According to Boeh, programming is hard, tedious work that very few people would choose to do for free. "The barrier of getting a game done is the size of the task. You have to write such a lot of code. If you have a wife and a child, it's hard to stay focused," says Boeh. While it's relatively easy to get a simple prototype up and running in a few weeks, months of tough and often frustrating work follow. The game must be polished; bugs must be found and eliminated; sloppy routines must be detected and smoothed

Fig. 2: Retro64's Cosmobots — proprietary shareware

out. These tasks are difficult and laborious - the sort of boring and repetitive tasks that most sensible people expect to be paid for performing. Boeh believes the value he adds to his software is his meticulous coding practices and polishing, and a few moments spent browsing the software library at *Retro64* is enough to demonstrate his superior craftsmanship and attention to detail.

Boeh feels he has good reasons for releasing his games under a proprietary, non-free license. Boeh's biggest fear is that unscrupulous competitors would use his code to quickly produce competing "knock-offs," or games that differ only superficially (if at all!) from his. Someone cloned his game *Z-Ball* and tried to fool consumers into thinking it was Boeh's game. A few developers even tried to copy Boeh's website, going so far as to clone his slogan "Where the fun is never old." Since Boeh is striving to earn a living doing what he loves - programming great games - he feels he has a good reason for keeping his code secret and doing his best to protect himself from competitors.

Boeh and Horn have made trade-offs. Horn has traded secrecy and a certain level of security in the hopes that he will be able to take advantage of the public's goodwill. Hundreds of developers have chipped in to help Horn find bugs and improve and extend his code. Boeh, on the other hand, has sacrificed this help for the sake of keeping his code secret and thus hopefully reducing the threat of competition. Which trade-off is more advantageous?

Although it is foolish to make a generalization based purely

on the experiences of two developers, in this case, it's clear that Boeh's method is earning him more revenue (I would *guess* he earns roughly $35,000 or more). Though Boeh didn't give me exact figures, he is proud to admit he earns a respectable living purely by producing and selling his games on the net. Horn, on the other hand, has made slightly over $200 selling CD versions of his game. For a developer faced with the choice of earning $35,000 vs. $200, the choice of whether to write free or proprietary software seems clear.

However, such figures are *highly* misleading. Horn is quick to point out that while $200 seems a paltry sum, the exposure the game has brought him amounts to much more. "It's helped me get a lot of jobs," says Horn. "I have worked a lot of places during the summer. I worked for Sony last summer, NVIDIA a few summers ago - I was working on OpenGL drive development." Producers looking for talent are impressed with Horn's work - not only because they like what they see in *VegaStrike*, but more importantly, they can get a good look at his coding practices. Source code is far better than resumes or recommendation letters for showing a potential employer that you have what it takes to contribute to important projects. After all, if you were a restaurant owner seeking a chef, wouldn't you want to watch that chef in action as well as taste her Chicken Roulade?

Another problem is that while Boeh has demonstrated his ability to produce quality games, he has not demonstrated his ability to work with large teams of other people - a critical skill in today's software development industry. Eric Raymond, author of *The Cathedral and the Bazaar,* puts it this way:

> "The developer who uses only his or her own brain in a closed project is going to fall behind the developer who knows how to create an open, evolutionary context in which feedback exploring the design space, code contributions, bug-spotting, and other improvements from hundreds (perhaps thousands) of people." (51)

Thus, releasing the source code to a piece of software not only allows potential employers to see a programmer's technical ability, but also her potential to manage a large project involving hundreds of other coders, artists, and musicians - a skill that sometimes seems more important today than programming.

Raymond also points out another vital characteristic of modern software development - very few programmers earn a living working for proprietary developers. The great majority of programmers work for "in-house" projects, creating and maintaining software for business and industries. Raymond advises his readers to check the want-ads of their local newspaper for evidence of this fact.

In short, a programmer striving to "learn the ropes" and get a leg up on the competition - the thousands of other aspiring programmers emerging from universities, colleges, and institutes - could do well for herself by developing and releasing a useful and influential free software program or contributing to an existing one. Raymond writes, "Prestige is a good way to attract attention and cooperation from others" which may very well earn the programmer much higher-paying jobs than she could otherwise expect (84).

The public benefits of free software

Though Bill Gates may sometimes contend that free software development is harmful to our way of life - even going so far as to refer to it casually as "communistic" in a recent CNET interview, the public benefits of public licensing are clear, and have been described quite compellingly by Lawrence Lessig, Richard Stallman, and Eric Raymond to name but a few.

Lawrence Lessig, author of several books that explore the great societal benefits of *commons*, makes one of the best cases for free software in his book *The Future of Ideas*. The idea is that a large pool of freely usable code forms a highly valuable and useful "commons" from which all programmers can take freely when building new programs. Developers who take advantage of this commons are freed from concerns about copyrights and patents, and from constantly having to re-invent the wheel. Lessig doesn't ever make the claim that *everything* should be shared in common. His point is rather that people should become aware of the great benefits to *all* by sharing certain types of resources. In Lessig's view, modern copyright and patent laws have become too powerful, giving powerful business and industry leaders an unfair advantage over the public. Perhaps a more compelling point though, is that these tyrannical practices are often injurious not only to the public but also to the industries themselves - it's hard to make progress or introduce innovation under the current regime.

Richard Stallman is more concerned about the *spiritual* effects that proprietary development has on programmers. In the *GNU Manifesto*, Stallman explains how the proprietary model allows programmers "to make more money, but... requires them to feel in conflict with other programmers in general, rather than feel as comrades." The secrecy and inability to share useful programs with their friends outside the company breeds a certain cynicism and pessimism that ultimately proves corrosive to a society in the "information age".

> Source code is far better than resumes or recommendation letters for showing a potential employer that you have what it takes to contribute to important projects

While Eric Raymond seems to prefer economic benefits of free software over philosophical concerns, he nevertheless shares Stallman's belief that free software allows for a more positive and fulfilling working environment than proprietary: "We're proving not only that we can do better software, but that joy is an asset" (60). One need only glance at some of the unpleasant news coming from *Electronic Arts* these days to be rest assured that free software is a blessed alternative to sacrificing one's principles for the sake of another man's profit.

I will end this section with an observation that may seem questionable at first: programmers have a unique responsibility to society and should think about their job in moral terms as well as economic. Programmers shouldn't be mere technicians doing unimportant, thankless drudgework. Rather, they should realize their critical importance and responsibility in a world gone digital. They are in many ways similar to the priests and monks of Europe's Dark Ages; they are the only ones with the training and insight to read and interpret the "scripture" of this age. Though powerful business leaders have tried hard to devalue their creativity and force them into working in demeaning conditions, programmers ought to recognize that *they* have the power to change this situation. No modern business or industry could survive without the diligent assistance of programmers. It's time that programmers became cognizant of this fact and used this leverage to make the world a better place - they have a responsibility not only to themselves, but to a future society in which their children will thrive or suffer. Freedom, openness, and sharing are not merely desirable; they are *essential* for the future of democracy and ensuring that our descendents emerge as *citizens*, not servants.

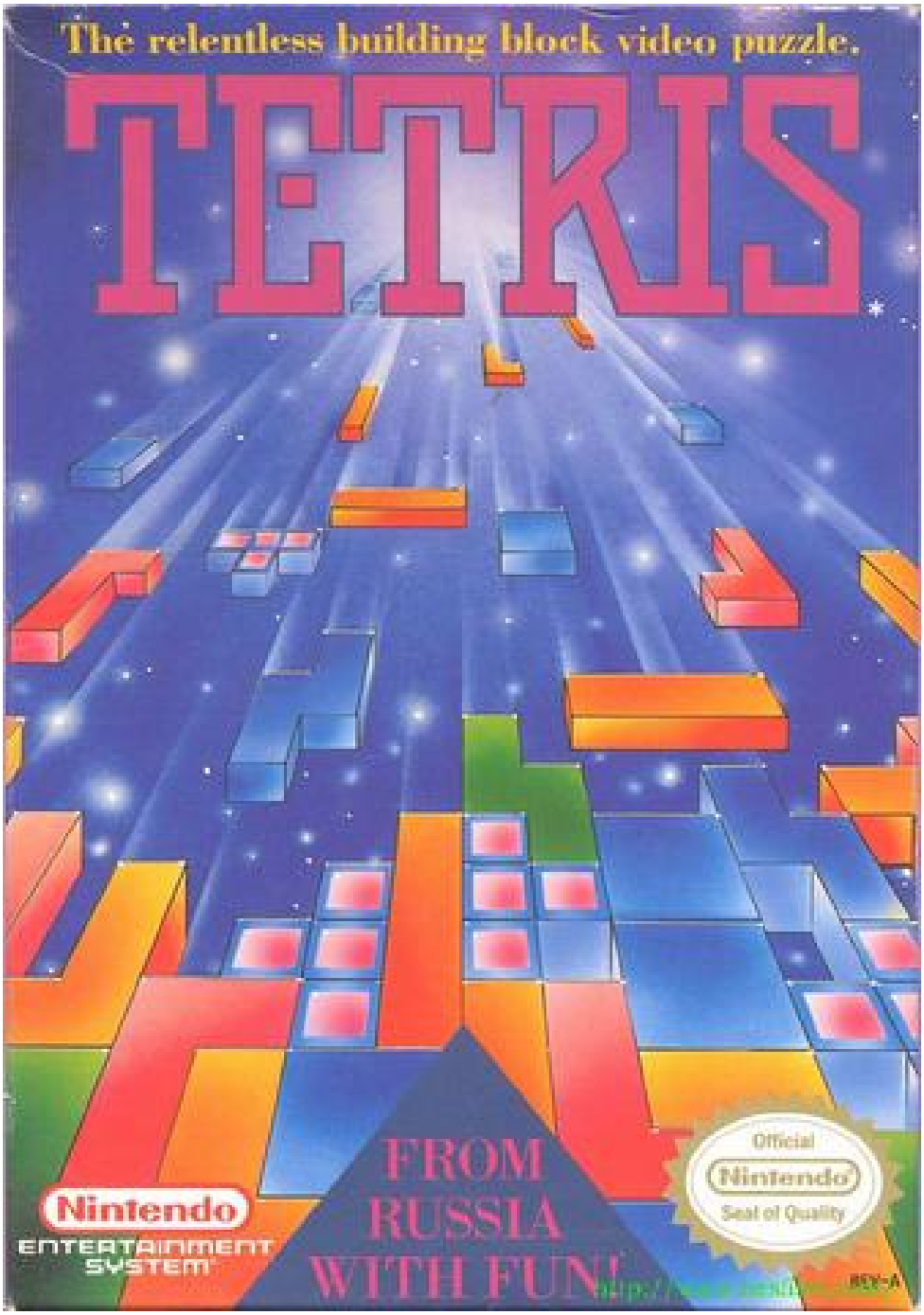
Platforms and permanence

I will finish this piece with a consideration that ought to appeal to any programmer's ego: *Permanence*. While most of the coding performed by programmers may be routine or mundane, other projects have a much different feel to them. Sometimes a programmer will be seized by an idea so exciting that it is difficult to keep her fingers still enough to enter the code. These are the landmark projects; the paradigm shifts; the software that we can only compare to true works of art. These works deserve our best efforts at preservation and are simply too valuable and precious to be controlled by any single entity or corporation. They are glorious gifts that should be rightly bestowed upon an eager and appreciative public, who will long remember the contribution and catapult the programmer's name to fame and history.

Let us take a handy example of such an event: Alexey Pajitnov's *Tetris* game, first released in 1985 in the Soviet Union. The game was so original and compelling that it clearly represented a breakthrough. Unfortunately, Pajitnov was living in a communist country that *supposedly* valued the public sharing of resources, *Tetris* represented too large a cash cow to be "sacrificed" to communist principles. The Soviet government claimed control of Pajitnov's game and made money by licensing it to publishers in other countries. The game sold extraordinarily well and made millions for the corporations that licensed and published it. The game made countless fortunes, but not for Pajitnov. Nevertheless, his name will likely live forever in history as one of the game industry's most influential innovators.

Let us assume that Pajitnov had lived in the United States in 1985 and had submitted his game to a commercial software publisher, such as the Nintendo Corporation. Would *Tetris* have had the impact it had if this had been the case? I very seriously doubt it, because, Nintendo would have undoubtedly been better able to leverage its "intellectual property rights" to generate more profit for itself at the expense of having the game reaching less players. It is likely that they would have taken a fist-of-iron approach to the hundreds of clone makers. Furthermore, the game would *only* be avail-

Fig. 3: Tetris Box – No mention of Pejitnov here!



able for Nintendo's own platforms (or licensed at exorbitant prices for computer software makers). Finally, if this had been the case, Pajitnov would not now have the rights to his game; those would have been assigned to Nintendo forever at the outset.

It's really anybody's guess whether *Tetris* would be as popular as it is today if Nintendo had been granted monopoly rights to its distribution. Several other NES originals, such as *Super Mario Bros.* and *Legend of Zelda,* remain popular today, and Nintendo has made them available for its newer platforms. Still, it's undeniable that even these games would be more accessible if Nintendo had released them into the public domain or under a public license. Of course, doing so would cost Nintendo some valuable "intellectual prop-

erty", but, then again, is that really a concern for the teams that created these games? I doubt most people in the street would be able to name a single person who assisted in their development - and what happens if Nintendo goes bankrupt (and its assets get tied up in a legal morass for decades) or decides not to release these titles on future hardware? Developers with heroic aspirations have to keep these possibilities in mind.

The situation is even more grim for developers for computer applications. Sure, Microsoft's *Windows* enjoys greater market share than GNU/Linux or other competitors. Nevertheless, even Bill Gates seems surprised at times that his corporation has achieved such great success and has held it for so long. Meanwhile, the United States government and plenty of foreign governments have taken Microsoft to court for monopolistic practices, and while Microsoft has endured, these attacks are unlikely to cease or grow less threatening. A developer who chooses to work strictly with Microsoft's own development software and proprietary tools must consider whether her projects - especially those "paradigm shifting" mentioned earlier - are really worth risking on a closed platform. Indeed, at this stage of the game, a true "killer app" for *GNU/Linux* would seem more likely to vault a programmer into the annals of history than a comparable application for *Windows*, where it would likely get lost in the sea of competing commercial applications. It is *certainly* true that an application, which threatened to significantly alter the way we use computers would seem a dangerous threat to an established corporation *whose future depends on maintaining the status quo.* Consider briefly how the proprietary software industry has responded to developments like peer-to-peer networking. Where did the majority of "killer apps" for the internet come from? Today, we know Tim Berners-Lee as the "inventor of the World Wide Web", but we scratch our heads when someone asks us who developed Apple's *HyperCard,* a stunningly original application that in many important ways was a progenitor of hypertext.

Bill Atkins must have anticipated this sad fate for his groundbreaking program when he insisted that Apple would release his program for free on *all* Macs. Apple chose to ignore this agreement when it released the next version of his program. What if Atkins had released *HyperCard* under a general public license? Or perhaps placed it into the public domain? Tim Berners-Lee was knighted by Queen Elizabeth in 2004. Who's Bill Atkins, again?

Concluding thoughts

The risks of writing proprietary software are many, and the sole benefit - quick cash - seems to pale in comparison to the many, longer-lasting benefits of writing free software. A truly wonderful program released under a free software license is much more likely to earn a developer prestige, reputation, influence, and fame than a comparable proprietary program. Besides these personal benefits, there are also societal benefits that are impossible to ignore by men and women of integrity. A developer intent on really making a difference ought to consider whether history shows that tyranny is superior to freedom; if feudalism is better than democracy. Surely, the history of the United States offers evidence that it is only when people are allowed *to be free* that they are also allowed to truly prosper. The same is visibly true of software. Freedom sells, and the future is buying.

Bibliography

[1] Lessig, Larry. *The Future of Ideas: The Fate of the Commons in a Connected World.* New York: Vintage Books, 2002.

[2] Raymond, Eric S. *The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*. Cambridge: O'Reilly, 2001.

[3] Stallman, Richard. "The GNU Manifesto." *Free Software, Free Society: Selected Essays of Richard M. Stallman.* Ed. Joshua Gay. Boston: GNU Press, 2002. 31-39.

Copyright information

© 2005 by Matt Barton
This article is made available under the "Attribution" Creative Commons License 2.0 available from http://creativecommons.org/licenses/by/2.0/.

About the author

Matt Barton is an educator and writer, who is currently living in Tampa, Florida. He is an advocate of free software and the Creative Commons. He hopes to receive his Ph.D. in May 2005.

ALL MUSIC AND ARTWORK IS
RELEASED UNDER CREATIVE COMMONS
ATTRIBUTION SHARE-ALIKE LICENSE

WWW.LIBRESOCIETY.ORG
WWW.CREATIVECOMMONS.ORG
WWW.WIKIPEDIA.ORG

E-LOCA-TRONICA 2005
STICK TO WHAT YOU DON'T KNOW

HTTP://WWW.LOCARECORDS.COM

INFO@LOCARECORDS.COM

YOU CAN DOWNLOAD MUSIC
VIDEOS AND TEXT BY VISITING
WWW.LOCARECORDS.COM

LOCA RECORDS, PO BOX 233,
BRIGHTON BN2 3WX, UK

The risk of using proprietary software

Do you know what you're feeding your computer?

Matt Barton

About one out of every 200 people is allergic to peanuts. Depending on the extremity of the allergy, a person suffering from peanut allergies who was accidentally exposed to peanuts might develop an itchy rash. Others might experience anaphylaxis, a severe reaction that can prove fatal. People who are allergic to peanuts have a tough time in America, where more and more foods are manufactured in factories that also process peanuts.

Thankfully, manufacturers and restaurants are coming under pressure to clearly label foods that either contain peanuts or were prepared with machinery that also processed peanuts. These measures have saved lives and helped Americans live healthier lives, because knowing what you're eating ought to be important to everyone. Without access to this information, hundreds of men, women, and children would die each year. People with a deadly sensitivity to peanuts would literally be playing a game of Russian roulette every time they tried a new food.

What if we lived in a nation where manufacturers weren't required to print the ingredients of their foods on their packages? What if food corporations had successfully lobbied Congress to allow them to keep their ingredients totally secret? We can easily imagine the arguments. "If we publish our ingredients, then our competitors will be able to duplicate our recipes". "We use chemicals and ingredients that might turn off consumers". "If people don't trust us, they shouldn't eat our food; we shouldn't be *forced* to list the ingredients". Nevertheless, in this case, common sense won out, and now we're entitled to not only read the basic ingredients but also get fairly reliable nutritional information about the foods we eat. We don't necessarily consider that the manufacturers are graciously offering us a service. Instead, we see this as our *right*, a demand that we reasonably make to manufacturers. If a manufacturer refused to tell us what is in its food, we would be stupid to eat it anyway. It's just *common sense* to require that manufacturers tell us what they're putting into our foods - because we put those foods *into our bodies.*

You have a *right* to know what's going into your food, you have a *right* to know what a piece of software is doing inside your computer

Now, let's consider another case that isn't really much different than food, but is nevertheless treated as though it *were*: computer software. No, I'm not saying that you should try munching your copy of *Half-Life 2*. What I mean is that software is something that you put in another type of body; namely, your personal computer. For the same reason that you have a *right* to know what's going into your food, you have a *right* to know what a piece of software is doing inside your computer. It ought to be *common sense* that software developers be required to publish this code for your review before you run their programs. When a software developer tells you, "No, just *trust us*," your mental

Fig. 1: Never has a game been so aptly named

red-alert should start sounding loud and clear. This is a dead giveaway that you should steer clear of this software and not even consider installing it on your machine. Sure, perhaps it is safe and legitimate. But how do you know? Is it worth taking a risk with all of your precious programs and data? Why should you even be asked to take this risk?

The propriety of trust

Let's explore this concept a bit. Let's say you are browsing the games at your local software shop and find a great new role-playing game from a major developer. Even though it's a bit pricey at $60, you're impressed with the description on the box and take it home. Unfortunately, after playing the game for a few hours, you decide you don't really like it. It's boring and not nearly as good as you thought it was going to be. Of course you can't take it back to the store since no one is going to trust you enough to believe you didn't make an illegal copy of the game. So, disgusted, you decide to use the game's uninstallation program to take this clunker off your hard drive.

This program deletes the entire contents of your hard drive. Gigabytes worth of papers, emails, family photos, and countless other valuable data is lost forever.

Yeah, right, you say. This would never happen. Yet it did.

The game is Stormfront Studio's *Pool of Radiance II: The Ruins of Myth Drannor,* distributed by Ubisoft and released in 2001 (see this IGN review (http://pc.ign.com/articles/162/162043p1.html) or this Game Over.net review (http://www.game-over.net/reviews.php?id=663\&page=reviews)). If you haven't heard of it, there's good reason. The game was one of the most pointless and sleep-inducing games since *E.T.* for the Atari 2600. In the world of big-budget commercial games, this isn't really anything unusual. It's also expected that there will be plenty of bugs, some of them show-stopping bugs, in early releases that will only be fixed later on by downloadable "patches" and "fixes." However, the development team responsible for *Pool of Radiance II* represents an all new low for proprietary development: The uninstallation script can actually damage vital system files and has reportedly wiped some users' hard drives completely.

Of course, the developers soon released a patch to replace the dangerous uninstall program, but is that enough to help us sleep better at night after installing a new proprietary program on our computer?

I started this article by describing why we, as a society, demand that food manufacturers tell us what they put in our food. What would be the equivalent practice we should demand of software developers? The answer is that they should release all of their source code so that we get a chance to *see* what their programs will do to our machines *before* we install and run them.

"Wait a minute," you say. "I don't know *anything* about software code. I'm *totally* code illiterate. How is that supposed to help me? I wouldn't be able to tell what the software was doing to my computer even if I had the source code!"

Well, you *could* learn to code. It's not impossible, and, in fact, not really more difficult than learning how to read or learning to speak another language. I'm of the opinion that everyone should learn at least the basics of programming. The rewards are immediate and immense.

But, let's say that you don't care and will *never* care about knowing how to program. Why would having access to the source code matter to you?

The answer is that while *you* may not understand the source code, there are plenty of other people who do. These people would be very likely to spot malevolent, dangerous, or just outright sloppy code coming from the developer and publish

their discoveries on the internet for all to see. If you read that a new game contained code that could delete random files on your hard drive, you'd know better than to install it. There are plenty of people out there who would happily perform this public service, and they would get something out of it, too. By having access to the code, they'd be able to learn quicker and faster how other programmers are working their magic.

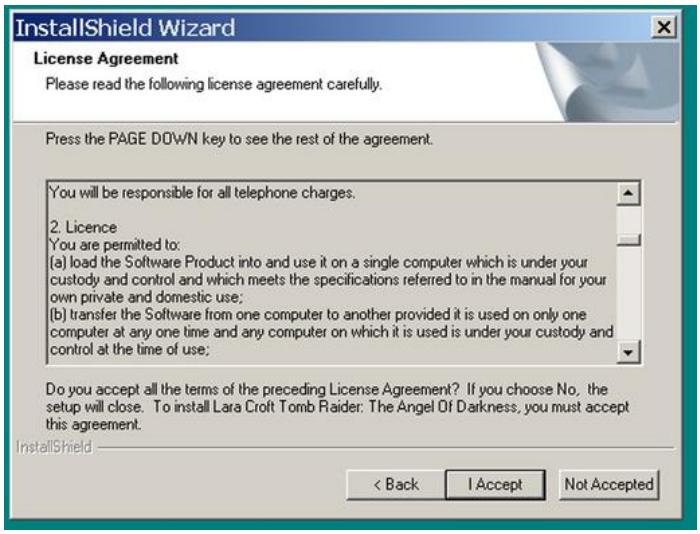
In short, proprietary developers depend on your naiveté - your ignorance and gullibility - to install and use their software. They say "Trust us," and that's supposed to be sufficient. If you don't like it, you don't have to buy their products. Now, keep in mind that the proprietary developers expect you to trust them with all of the programs and data on your computer - but do they trust you?

Quite the contrary. In fact, most commercial developers have taken it upon themselves to include programs that prevent you from making a legitimate backup copy of the software products you purchase. They've also made your life more difficult by asking you to enter long and complicated registration or verification codes into your software before it will install on your drive. Many games require that you find and keep the CD-ROM inside your drive anytime you want to play it. Other copy protection schemes manipulate your system so badly they may trigger your anti-virus protection programs! There are even reports that some copy protection programs scan your drive and will not allow a program to be installed unless you remove certain programs first. Should I mention other "features" like region-encoding? Do you enjoy having to click "I ACCEPT" to pages of incomprehensible legalese before installing a product you just paid $60 or more to own?

No, the proprietary software industry doesn't trust us. In fact, it is so suspicious that it is willing to compromise the efficiency and convenience of its programs in a vain attempt to thwart hackers from making and distributing illegal copies. They don't even consider you the "owner" of the software you purchased; it's just "licensed" to you. If you don't know the difference, don't worry - the developer's attorneys will be happy to explain it to you in court.

Let's put things in perspective. When you buy a proprietary program, you're expected to just have faith that it won't damage or destroy your computer. On the other hand, the makers of the program don't trust you one bit. In fact, they take every possible precaution to restrict your freedom and

Fig. 2: A typical end-user agreement. No, you won't be negotiating the terms of this "agreement"


InstallShield Wizard
License Agreement
Please read the following license agreement carefully.
Press the PAGE DOWN key to see the rest of the agreement.
You will be responsible for all telephone charges.
2. Licence
You are permitted to:
(a) load the Software Product into and use it on a single computer which is under your custody and control and which meets the specifications referred to in the manual for your own private and domestic use;
(b) transfer the Software from one computer to another provided it is used on only one computer at any one time and any computer on which it is used is under your custody and control at the time of use;
Do you accept all the terms of the preceding License Agreement? If you choose No, the setup will close. To install Lara Croft Tomb Raider: The Angel Of Darkness, you must accept this agreement.
InstallShield
< Back
I Accept
Not Accepted

guarantee that you'll obey the rules they establish to regulate your behavior.

Does this sound like a good deal to you? Of course not. But is there any alternative? After all, we have to have software to run on our computers if we want to get any work done or have any fun with them.

The free software alternative

The good news is that there is a wonderful alternative to proprietary software. You guessed it: free software, and it's becoming more plentiful and effective every day. Now, don't get the wrong idea: free software isn't necessarily free as in it doesn't cost you anything. Folks who work long and tedious hours developing great software have a right to ask for compensation and often do. When people talk about "free software," they're not describing software in the public domain, which *is* free and doesn't cost anything. Free software is simply software that has been released under a *public license,* such as the General Public License (or the GPL). If this is confusing, just think about what we mean when we say "free speech." This doesn't mean that you go the library and start stealing books. It means that somebody can't use the government to force you to shut up or not print something just because they disagree with your opinion. You're free to use it, copy it, study it, and improve it however you

see fit - you don't have to answer to anybody, even the people that developed and released the program.

Let's put this in context. Assume that you want to use a software program that will make your screen flash "Microsoft sucks!" over and over again. Now, if this screensaver program is proprietary, the owner of that software can tell you that your use for that program isn't appropriate, and represents a violation of your "End User Agreement". In fact, if you don't stop, you're going to be sued. You aren't free to use this program in a way that the developer doesn't approve of. It's a *non-free program*. In fact, instead of just seeing "I Agree" when presented with end-user agreements, I think you should see "Yes, Master". This terminology would much better reflect the type of relationship you are entering when you install a proprietary software program. On the other hand, if your screensaver is a free software program, it doesn't matter what the developer says. If she happens to see you running the program and calls to ask you to change your message, you can tell her where to shove it. However, I doubt very seriously such a preposterous thing would ever happen, because free software developers would be the first people to tell you that you have a right to say whatever you want.

Free software also has another huge advantage: the source code is always available for your review. If you want, you can even compile it yourself so that you know *for sure* that it isn't doing anything questionable to your data. If you decide that you don't like the way it operates, you can either fix it or find someone who can. Of course, you may have to pay this person to make the changes you need, but it's an option that you don't have if you're using proprietary software.

Conclusion

Some people like to downplay the importance of having free software. They say, "Well, if the program does what you need it do, it doesn't matter if it's free or proprietary". Bill Gates has taken to referring to free software developers as communists, hell-bent on undermining democracy. Such critics always manage to make their position sound like pure common sense, even though what they're preaching is the opposite of common sense. No sane person would eat whatever was handed him without bothering to find out what it was. No sane person would claim that demanding that we hold manufacturers responsible for their products is "communist". Indeed, it's far more "communist" to believe that we should just let other people make these decisions for us; that it isn't really our business and that we should just trust in our leaders to do what's right.

I believe that people are becoming more aware of the dangers posed by proprietary software developers and are getting tired of playing their game. We have a right to know what we're putting into our computers, and even if we can't read code, we know that other people can and will tell us if something is seriously wrong. We're also getting fed up with developers who don't trust us enough to let us make backup copies of the programs we purchase, yet expect us to trust them with the precious data stored on our computers. Finally, we're sick of being told how to use the programs that we buy and that we're not allowed to change them if we want.

You *do* have a choice, but that choice is not likely to come from the big companies that have been making a fortune selling you proprietary software and dictating how you should use it. When you decide that you've finally had enough, then it's time to learn about GNU/Linux and the thousands of free software alternatives that are just as good (if not better) than their proprietary equivalents. If you're totally new, a good place to start is KNOPPIX or SIMPLY MEPIS. These free operating systems are easy to install and fully functional. You've tried tyranny; now see how you like freedom.

Copyright information

© 2005 by Matt Barton

This article is made available under the "Attribution" Creative Commons License 2.0 available from http://creativecommons.org/licenses/by/2.0/.

About the author

Matt Barton is an educator and writer who is currently living in Tampa, Florida. He is an advocate of free software and the Creative Commons. He hopes to receive his Ph.D. in May 2005.

Finding alternatives in developing software

Comparing free development tools and libraries with proprietary ones

Martin C Brown

Developing software within the free software model can be achieved with all sorts of different tools, but choosing the right tools can make a big difference to the success of your project. Even if you are developing a proprietary solution, there are benefits to using free software tools to achieve it. But what free software tools are available? In this article I'm going to look at the development tools available, from languages and libraries to development environments, as well as examining the issues surrounding the use of free software tools by comparison to their proprietary equivalents.

Development languages

The languages used in your development are generally free. Despite attempts by companies to protect their language specifics from the development community, the majority of those used today fit into the free software category. A lot of these languages are free not necessarily because they were outwardly designed that way but because, like human vocal languages, there was a need for an agreed set of standards.

> Choosing the right tools can make a big difference to the success of your project

Unlike the other components in the development process, the language you use must have a very specific set of semantics. This is because, as the basis and groundwork for your applications, there must be a commonly agreed structure. Now imagine if everybody that supposedly used C used a slightly different format for specifying a function. Essentially the structure of the language wouldn't be standard, and if the structure wasn't standard then we'd end up with thousands of different languages that might differ apart from the use of a semicolon to terminate lines.

Because of the need for standards, specific languages have specific rule sets and by their very nature these rule sets need to be published for people to program in that language. Ultimately, this means that most languages are technically free software or there is a free software solution for that language available.

Returning to the topic of the available free software languages, off the top of my head, I could probably list 15 commonly used languages that are essentially free, starting with C. C was originally developed as part of the Unix development process and the design specifics and semantics of the language were published; thus we have an open standard and one of the languages with the largest user bases and associated tools, libraries and environments. Most operating systems and their components are written in C and that includes both free software solutions like BSD and Linux and commercial offerings such as Solaris and Windows. This has the benefit that it's possible to easily extend the operating system by writing more C applications without the need for brand new libraries for basic functionality. Even more ironic is the fact that most other languages now tend to also be written in C and, in most cases, the libraries for those

Fig. 1: The Mono Project, .NET without Microsoft

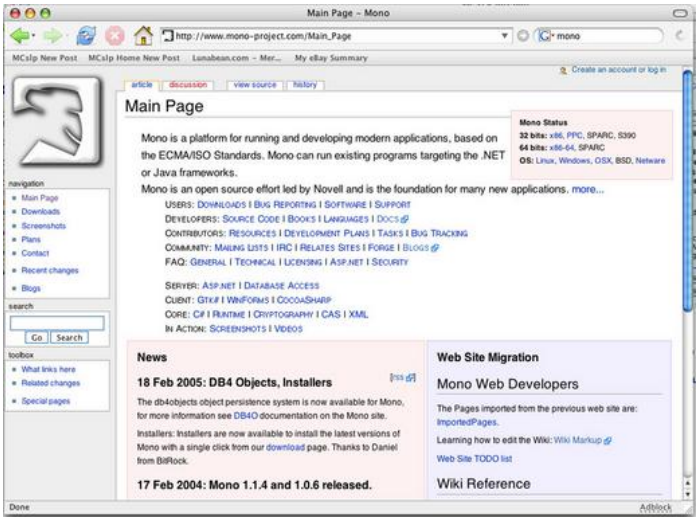
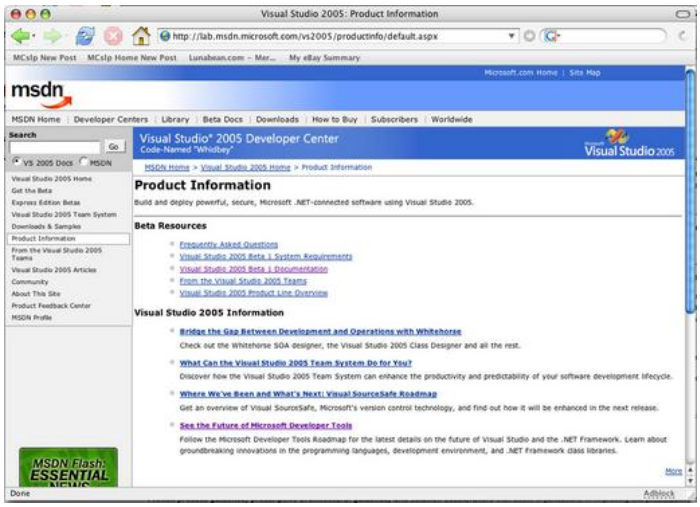

Main Page - Mono
Main Page
Mono is a platform for running and developing modern applications, based on the ECMA/ISO Standards. Mono can run existing programs targeting the .NET or Java frameworks.
Mono is an open source effort led by Novell and is the foundation for many new applications. more...
News
18 Feb 2005: DB4 Objects, Installers
17 Feb 2004: Mono 1.1.4 and 1.0.6 released.
Web Site Migration
Mono Web Developers
Wiki Reference

Fig. 2: Visual Studio .NET 2005 (Whidbey), currently in beta



languages ultimately interface to the C equivalents within the operating system.

Other languages which are free including C++, Perl, Python, PHP, Ruby, Pascal, Modula-2, ML, Fortran, Cobol, Lisp, Smalltalk, Tcl, awk/gawk and many others.

Now there are a couple of notable languages which I haven't mentioned, specifically Java and the 'sharp' series from Microsoft, including C# and J#. The first, Java, has free software compilers available for it (notably gcc), but the language itself is a proprietary project of Sun Microsystems. If you want to make full use of the Java language you really need the full Java libraries and associated compatibility, and that only comes from the Java runtime and development environment, which comes from Sun.

Languages which are free including C++, Perl, Python, PHP, Ruby, Pascal, Modula-2, ML, Fortran, Cobol, Lisp, Smalltalk, Tcl, awk/gawk and many others

Similarly, C# and J# are projects developed and managed by Microsoft. Both are object-oriented languages based on C, C++ and Java (and I really do mean a combination of all three, to varying degrees, in each case), which are specially designed to work with the .NET environment.

Like Java there are free software solutions available; the Mono project have developed a C# compiler and an almost complete .NET library suite which enable you to write, build and deploy C#/.NET applications on Unix and other platforms. But, C# and J# are proprietary languages with the development effort driven by Microsoft.

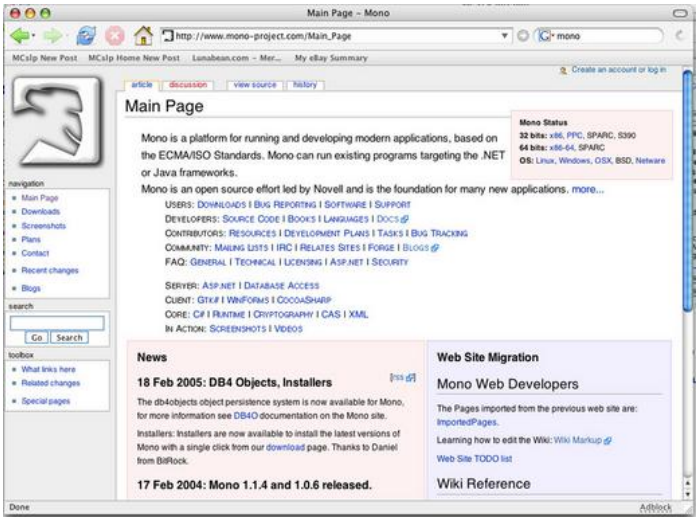
As you can see, on the whole the development language - the core of any development process - is a free software component, largely by its very design and existence. Now let's move on and see the effects of other components within the development process.

Development tools

One of the key reasons I write and develop free software is because of the tools and environments available to me. After 15 years as a developer, 12 of them professionally, I still use the same environment I've always used; emacs, make or Ant and whatever compilers or tools I need such as C, Perl or Java.

This might seem a little archaic compared to some of the tools that are available; for example when developing under Windows surely the best choice is one of the various integrated development environments (IDEs) such as Microsoft's Visual Studio. The problem with these tools is that they expect you to work in a specific way and they often tie you down to that way of working. Within Visual Studio for example you have to use their editor, particularly if you want to use features like code completion and expan-

Fig. 3: The Eclipse IDE in action


Main Page - Mono
Main Page
Mono is a platform for running and developing modern applications, based on the ECMA/ISO Standards. Mono can run existing programs targeting the .NET or Java frameworks.
Mono is an open source effort led by Novell and is the foundation for many new applications. more...
News
18 Feb 2005: DB4 Objects, Installers
17 Feb 2004: Mono 1.1.4 and 1.0.6 released.
Web Site Migration
Mono Web Developers
Wiki Reference

sion. Visual Studio also expects you to use the libraries and tools in the .NET framework for your development, and it also provides its own build and testing environment. All of these items become constricting components, designed to keep you using the IDE for all of your development and therefore allowing the proprietary developers to force you down a particular development and deployment route.

The benefit of emacs is that it's an environment I'm familiar with, one which is supported on multiple platforms and one which, with a little more command-line work than others, provides a flexible solution irrespective of the language, and more importantly, the platform on which I am working.

For those that want IDEs there are numerous potential choices available in the free software space. Although emacs is not best known as an IDE, it actually has most of the IDE components available, including project management, source control and building tools. For many of these components, the functionality is actually provided by another free software tool. For example, when building an application you could use GNU make, source control can be managed by the Concurrent Versioning System (CVS) or Subversion and the actual compilation can be processed by gcc.

IBM tried an interesting approach to the idea of proprietary development tools and turned it into one of the best known and acknowledged free software development environment. The Eclipse project was designed by IBM as their new IDE and was set to replace their existing development environments. Eclipse is entirely designed on the basis of plug-ins. There is a very minimal "kernel" to the Eclipse platform, everything else is essentially a plug-in to this kernel, extending and expanding the functionality of the application as it goes. The major benefit of the extensible architecture is that it makes it easy for a developer to modify and adapt the platform to work the way he or she likes. It also means that Eclipse can be used to develop any application for any language and environment. This eliminates one of the major complaints about most proprietary development platforms.

The libraries you use in your application are as important as the application itself. One of the key issues with libraries is that they will affect where your applications can be used and displayed

At a cost of US$30 million, Eclipse didn't come cheap but the result was an incredibly flexible and extensible environment. After the initial development, Eclipse was released to the community and is now developed in the same manner as other free software projects like emacs, Linux and GCC. IBM still use Eclipse as the basis for their commercial development applications - the new Rational Software Development Platform is based on the Eclipse environment and the IDE that is used within the WebSphere development suite (WebSphere Studio Application Developer) is an Eclipse application. Both suites make heavy use of the plug-in architecture of the Eclipse environment to add additional functionality and value to the products.

Eclipse is not the ultimate utopia. Ironically for a system that has been designed to be so flexible and open, it is actually written in Java. Also, Eclipse works on the basis of specific projects and workspaces for controlling the development process.

Development libraries

The libraries you use in your application are as important as the application itself. One of the key issues with libraries is that they will affect where your applications can be used and displayed. This is the issue of portability, and I'll get on to it later in this article.

The other issue is that they affect development from the point of view of the functionality of your application. Most

Fig. 4: The Simple DirectMedia Layer

proprietary libraries are developed with very specific goals and ideals in mind, often to fit the interests of the developers in question. For example, graphics libraries are often developed with the ideals of the graphics hardware in mind, rather than providing an easy to use and flexible graphics library for general use. Using a proprietary library in this situation leads to the development of an application that only works with very specific hardware and that in turn leads to a closed and limited application.

There are, in fact, a wide range of free software libraries available that cover just about every avenue you could want. For example, for multimedia the key technology on Windows would be DirectX, a proprietary solution from Microsoft that provides full multimedia capabilities, from playing music and video to displaying 3D graphics for games. DirectX is only supported on the Microsoft Windows platform but does have the advantage of providing an abstraction layer to a wide variety of underlying hardware.

The Simple DirectMedia Layer (SDL) library provides similar functionality, but does so through a free software model and provides the same functionality across a wide range of platforms. You can write an SDL application that operates on a number of platforms with no changes to the code; something impossible with a proprietary solution such as DirectX.

Better still - and this particularly applies to libraries like no other part of the development process - if there is some element of the library which does not fit in with your plans but which you think would be useful to others you can expand and add the functionality and make it available to others.

Other free software libraries, that can be used in place of proprietary equivalents, also exist. At the basic level, the GNU C library (glibc) provides the core functionality for the C language and libstdc++ is the standard C++ library.

There are database libraries, for example Berkeley DB in place of Microsoft's Jet and numerous networking libraries in place of the proprietary networking solutions.

Portability

I touched briefly upon this subject earlier in regard to the potential for lock-in with libraries. Surprisingly few companies think about this when developing their software, and then get stuck when the vendor drops support for the library. Developers relying on proprietary solutions also find that the lack of flexibility becomes a problem.

Furthermore, the effect can be felt elsewhere in the development process. For example, develop an application using Visual Studio .NET and the chances are that porting your application to a different development environment will be incredibly difficult, because all of your code will be designed to work within the .NET framework and use functions and facilities only available to your .NET developed applications.

Worse still, your application not only becomes locked-in to your development environment, it also, ultimately, becomes locked into your development platform. Creating an easily portable piece of code that can be used on Windows and, say, Mac OS X becomes a mammoth task. There are obvious differences between the two platforms, but the common elements within an application clearly remain the same.

Take everybody's favourite proprietary application, Microsoft Word. There are versions available for Microsoft Windows and Mac OS X and the two applications on the two platforms share about 90% of the same functionality (with the rest being taken up by differences like Entourage and the obvious interface specifics). Now you can't develop Mac OS X applications within Visual Studio .NET, but migrating what is core code between the two platforms can't be an easy task.

As a developer, how do you get round this?

Well, using free software is an obvious answer. In general, free software solutions don't lock you in to a particular style

of working and they will be based on open standards and interoperability. If you have developed an application within the GNU framework, for example, then portability is a key part of the development process. GNU tools like autoconf, configure, make and gcc exist and are used for the purposes of making software available on as many platforms as possible.

If you are a developing a free software project then using free software tools is vital because of the ease with which you will be able to share information and the project itself. Imagine trying to get everybody to contribute to a project if they had to obtain a closed set of tools to do it. The chances of people getting reliably involved are slim, and probably non-existent. The more people you can get to aid in the development of your project, the better it will become; and using other freely available tools is the way to do it.

Spotting a free software development environment

I should highlight the fact that not all development environments were created equal, even free software ones, and the specifics of the environments can be difficult to spot. For an excellent example of this look no further than Mac OS X.

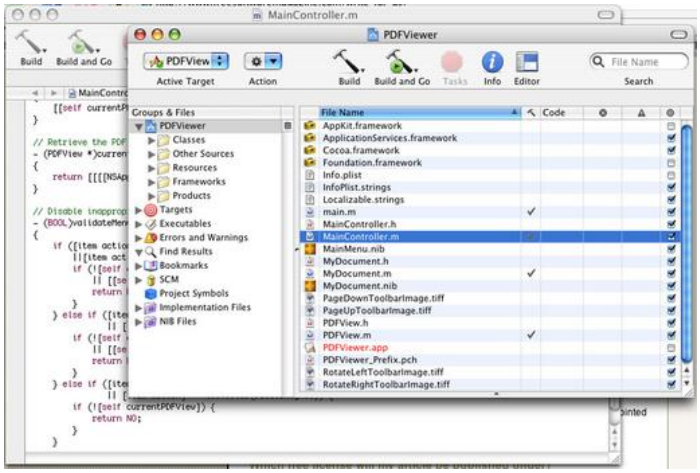
At a fundamental level, Mac OS X is a proprietary operating system based on a free software platform, and its development environment is based on a free software platform, supported by a proprietary development tool. Confused?

Well, Mac OS X is basically a combination of the Darwin operating system and the Cocoa windowing environment that gives OS X its look and feel. Darwin is based on the BSD operating system, a free software project and one of the oldest.

The main development environment on Mac OS X is Xcode. While Xcode is a proprietary IDE designed to work only on the Mac OS X operating system, its underlying code and toolsets are based on the GNU suite. Your applications are compiled using gcc, linked using GNU ld and rely on some of the libraries provided by the Free Software Foundation.

What we have here is a situation where you can develop software, for either a proprietary environment or a generic Unix environment, using a proprietary IDE that builds and compiles applications using a free software development environment.

Fig. 5: Xcode, the proprietary IDE based on free software tools



Does that make Xcode a proprietary solution? Unfortunately yes, but the key is, that underlying this, is the fundamental use of free software tools and therefore a resounding endorsement of the free software development ideals.

So how do you spot a free software development environment? Plain and simple: research. The chances are, if a development environment uses free software tools then it will shout about it. If it doesn't, it's probably closed and proprietary.

Copyright information

© 2005 by Martin C Brown
Verbatim copying and distribution of this entire article is permitted in any medium without royalty provided this notice is preserved.

About the author

Martin "MC" Brown is a freelance writer and consultant, he works with Microsoft as an SME, is the LAMP Technologies Editor for LinuxWorld magazine, a founding member of AnswerSquad.com, Technical Director of Foodware.net, and has written books on topics as diverse as Microsoft Certification, iMacs, and free software programming.

Freelock Computing

The Open Source for Business Solutions

Open Source is about solving problems. We solve business problems for small and growing businesses in the Puget Sound area. We provide technology strategy consulting, implementation, and hosting. We work with proven open source software. Go to freelock.com for details.

"In today's world of crappy software vendors who provide crappy products and next to zero service at premium prices, it's refreshing to work with someone who is honest, thorough, reasonable and willing to do what it takes to meet the customer's needs."

– Eric Leung, IS Manager, Outdoor Research, Inc.

We Wrote The Book!

"'Open Source Solutions for Small Business Problems' is a very highly recommended book for anyone who is looking at the open source market ... This is easily one of the best Linux books of the year; providing a management level view of the Linux world without the technical focus of other books."

– Harold McFarland, Readers Preference Reviews

Find an open source service provider or sign up as one!
http://opensourceproviders.com

Contact us at:
info@freelock.com
1-206-579-4836

Freelock LLC
www.freelock.com

A server for education

Wims is a “magic” server

Georges Khaznadar

I recently encountered a group of very enthusiastic teachers, who wanted to convince me to try a new e-learning environment, with astonishing quizzes, and drills of extreme originality. However, as I’d last used computers in the seventies, I was initially sceptical. Back then computers had just been used to send humans to the Moon. It was hard for me to make the leap from those machines to the machines of today. And quizzes seemed to be a strange use of such a powerful resource.

Nevertheless, I found quizzes to be interesting when they are randomly generated from huge question and answer databases. Wims can do that, and other e-learning systems can do it too.

But Wims can do more: it comes with state-of-the-art syntax analysers, which understand a variety of specialised languages, which enables the server to deal with open answers to open questions.

The price of e-learning

Computer-aided learning is a good solution for many students, particularly when they need to repeat the same training again and again. Present software tools allow you to design attractive interactions, which emphasize clearly, concepts that are difficult to raise with other more traditional tools.

Have you ever tried to author an interactive exercise for your students? If so you may have found that you worked for an entire afternoon to create an interaction lasting ten minutes for the average student. So the throughput is about 25:1 (25 times ten minutes of work to produce an interaction lasting 10 minutes). This throughput ratio can be bigger or smaller, depending on your ambition and the complexity of the interactive sequence. If your ambition is to produce it as a TV show, a throughput of 30,000:1 would not be surprising.

A Wims Virtual Class can contain worksheets, course documents and exams. It has a forum featuring a rendering engine for algebraic expressions

Now, what if your next class is tomorrow? How can you author an interesting sequence in such a short time? That’s where Wims comes in. It uses powerful generators to translate an educational intention into readily usable interactions. This article explains how it works, and why it’s not possible for so many powerful applications to be packed in a single widely distributed proprietary product.

Wims as an exercise server

Ever found an interesting server for educational exercises? Not just drills, and quizzes, etc. I mean something really interesting, something you want to use for more than a few minutes. If you have, Wims is a better one. Go to the

Wims site (http://wims.unice.fr/wims), and have a look at these examples:

Tab. 1: Some examples

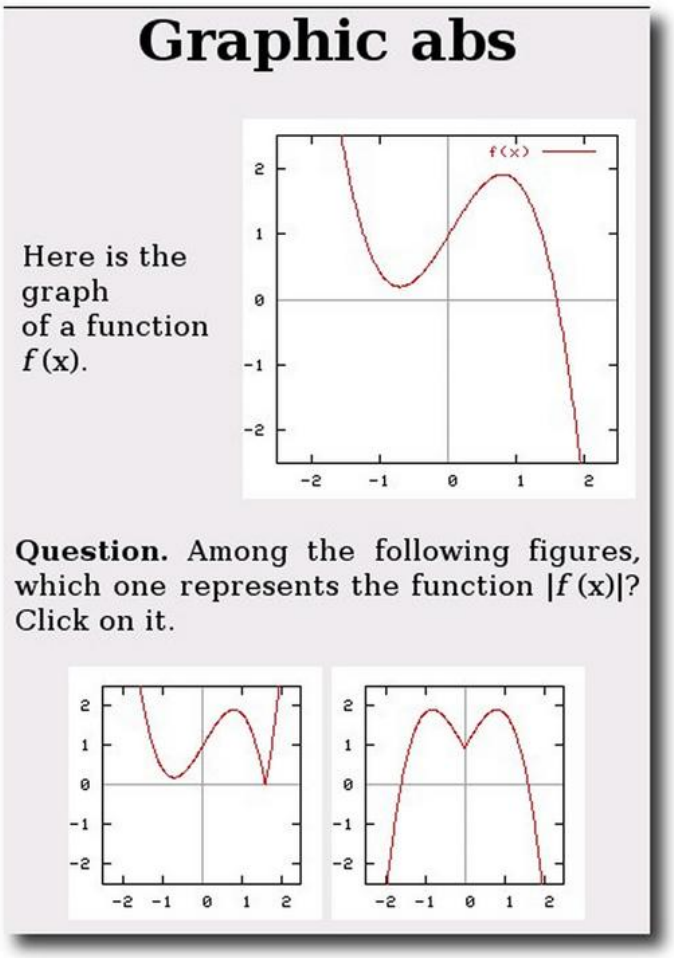
Example for the domain...	Keywords for the search engine
Interactive geometry	"triangular" (select the first hit)
Elementary arithmetic training	"arithmetic table" (select the first hit)
Algebra, at a higher level	"gauss" (select the first hit)

You can either access the site directly or by going to one of the mirror sites, see the link "mirrors" in the upper part of the main page.

Two students... collaborate?

Let's imagine two students who are in neighbouring seats, each with their own computer. They are trying to get a good score in a module dedicated to absolute values in maths.
They are given exactly the same exercise.
As the challenge is important, Dean asks Clive: "Where should I click?" Clive considers his neighbour's display, and says: "Click left". So Dean understands and gets a good first score. Unfortunately, the teacher configured the exercise to ask the same question many times. As the second figure appears, Dean asks Clive "Where should I click?" and gets the same answer: "Click left"... So now Dean is sure to be on the right path, and when the next question comes along, he clicks left without asking, and again it's the correct answer. Unfortunately for Dean the correct answer for the fourth question is not the left hand figure. When Dean shouts "Oh what a stupid exercise!" Clive considers the display, and says "Dean, don't you know? An absolute value must always be positive!"
Now let's consider the situation: after a few seconds, the two students come to make a verbal exchange at a very high level:
"An absolute value must always be positive!" shows a mathematical rule, which is a highly cognitive object. Clive does half of the teacher's work.
A little later, Dean might ask more questions, but organising a racket to steal useful answers from clever students is impossible: even clever students are forced to study each individual case before giving an answer. Communicating knowledge at a high level is the only possible way.

Fig. 1: Here is what Dean sees

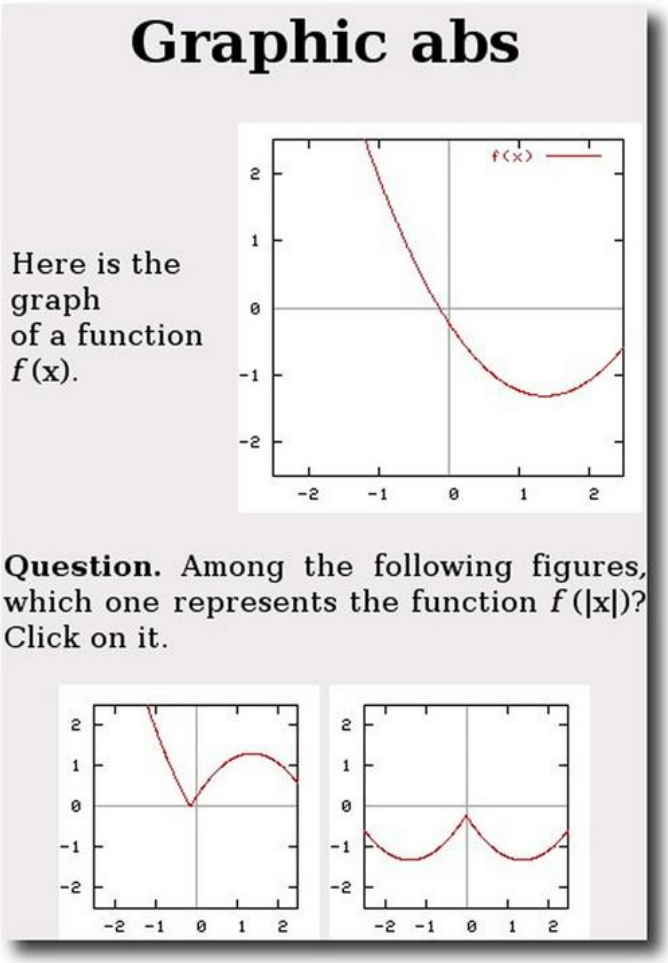

Graphic abs
Here is the graph of a function f(x).
f(x)
Question. Among the following figures, which one represents the function |f(x)|? Click on it.

When you get under the Wims hood, you discover powerful engines

Wims is built on top of a Unix or GNU/Linux system, which favours communication between processes.
The official mirrors of Wims currently use the following engines:

- Maxima, a Computer Algebraic System which is often compared with proprietary programs like Maple and Mathematica.
- Pari-GP, yet another Computer Algebraic System. Its speciality is the theory of numbers, polynomials and rational fractions.

Fig. 2: Here is what Clive sees


Graphic abs
Here is the graph of a function f(x).
f(x)
Question. Among the following figures, which one represents the function f(|x|)? Click on it.

- Gap, a Computer Algebraic System specialised in the group theory.
- Gnuplot, for rendering 2D and 3D plots
- Imagemagick, which enables converting series of images to animations
- Povray, to render algebraic surfaces by ray-tracing
- Chemeq, a converter of flat chemical notations to LaTeX, which can perform various verifications.
- TeX, to render algebraic formula (will be backed up by a MathML generator soon)
- Units-filter, which parses the physical quantities.
- Flydraw, a quick and efficient tool to create dynamic images.

However Wims is not limited to this rich set of applications: you can add every other application able to communicate with Wims. The only requirements are to be able to get parameters in the environment string, and to output either text to the standard output or data in a particular file. For example, graphics have to be output as files named insert1.png, insert2.png, etc. Imagemagick allows you to deal with a variety of graphic formats, including JPEG, GIF, animated GIF, PNG, and MNG.

How can it be so powerful?

Here we reach the main point of this article: how can so much wealth be contained in one product, which can be run even on more modest configurations? If you're searching for a CAS (Computer Algebraic System) for your students, there is nothing cheaper than $100. How is it possible to have the same thing on-line, with more features, open to thousands of students at the same time?

WIMS is a Magic Server. That's because Wims is free software, using existing free software programs.

Let's consider the proprietary way. Very few companies can afford to control programs of such varying specialities as graphics, mathematics, physics, chemistry, and so on, at the same time. A product gathering this many state-of-the-art applications covering such a variety of domains would imply expensive agreements between different companies, each having to make profit, and concerned by the possibility of diffusion of its knowledge. With such rules, complex software products often become more expensive than the sum of their component parts.

Now let's consider Wims: it contains a glue engine, able to integrate any application under Unix or GNU-Linux. It is linked to the independent programs, each of which is written by specialists. The current set of components for this glue engine totals roughly 3 MB, whereas the satellite applications sometimes three times as large. With the most powerful satellite applications, no change was made to the code. The availability of the source code makes it possible to write wrappers that ensure correct collaboration. Adding a new feature to Wims is just a matter of shaping a new glue component, which can be very simple.

You can use loads of pre-developed exercises

You can open a new Virtual Class for your students and assign them worksheets, in a matter of minutes. First find a Wims mirror near you: every Wims site has a link to official

mirrors, and the first web site on the list, managed by the author of Wims, Gang XIAO, may be less responsive, particularly when the students of the University of Nice (France) have an exam.

Then follow the link to the "teacher's area", and another link to create your class. You fill in a form with your name and your e-mail address, you then choose passwords for you and for your class, and you will be given control of a new Virtual Class: just watch your mailbox.

Once your class has been created, you can assign worksheets to your students: a worksheet is a collection of exercises picked in the pool of exercises from the web site. Most of the exercises are configurable, and you can configure the scoring features (severity, importance of the questions, etc). Then you assign the worksheets to your students, who can access them after an authentication step. You can create the students' accounts yourself, or let your students self-subscribe (they will need the password of the class, not your personal password).

You can also add exercises of your own, created by the easy authoring interface. A Virtual Class features Course documents, easy to link to exercises or interactive demonstrations, worksheets can be used as exams: then strong anti-cheating mechanisms are activated.

Creating a worksheet for your students

Enter a virtual class that you have created prior. Once you are authenticated, you enter the main page of the class, and there is a link to create new worksheets. Give it a title and a description, then add exercises you require by cycling through the following steps:

1. Go to main page of the class, and use the search engine to locate relevant exercises.
2. Follow a link given by the search engine, configure the exercise (qualitative and quantitative attributes), and test the exercise.
3. Once the exercise conforms to your requirements, put it in your worksheet (use the link at the bottom of the exercise to insert it).
4. Configure the subtitle of the exercise, the required score (so students must repeat the exercise to reach the score), the weight of the exercise in the worksheet.

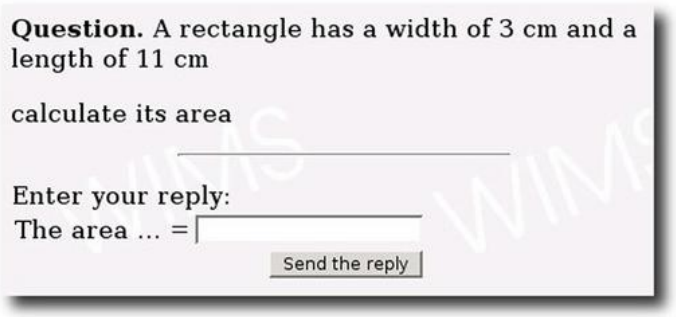
Create a new exercise

Wims new exercises can be authored in two formats: the Modtool format, which gives access to any feature of the Wims engine; and the OEF (Open Exercise Format) format, featuring less flexibility, but very easy to use. The OEF format has powerful primitives, which make sense to teachers: \statement, \choice, \reply, \step, etc.

There is also an assisted composer for the OEF format, which is usable on-line, it's the Wims module Createxo (follow the link "simple interactive exercises" at the bottom of the main page of each Wims server).

My first OEF exercise

Fig. 3: My first exercise in action, after submission



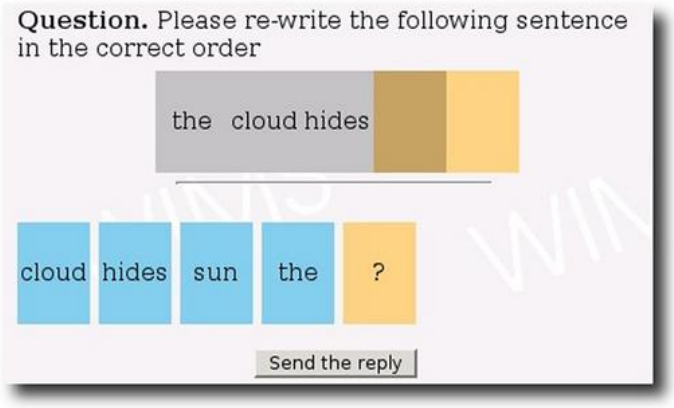
Type the following text (or better, just copy and paste it) into the on-line facility for uploading OEF exercises on this page (http://wims.unice.fr/wims/wims.cgi?module=adm%2Fcreatexo) (then click on the link "raw mode" to be given a text area to paste the source).

Here is the OEF source:

```
\title{My first OEF exercise}
\author{Clever Cleverer}
\email{clever@ofset.org}
\license{GPL V.2}

\integer{x1=random(1..9)}
\integer{x2=random(10..19)}
\integer{prod=\x1*\x2}

\statement{A rectangle has a width of \x1 cm and a
length of \x2 cm
calculate its area}

\reply{The area \ldots{}}{\prod cm^2}{type=units}
```

Wims for non-scientific topics

Fig. 4: The second exercise in action



Here is the template source for a very simple exercise based on ordering a phrase.

There is very little customization required, just modify the lines 5, 6 and 7. (This template comes thanks to Benoît Markey)

```
1:  \title{Template clickfill}
2:  \language{en}
3:  \author{MARKEY Benoit}
4:  \license{GPL V.2}
5:  \text{phrase1 = the,cat,eats,the,mouse;
6:  the,cloud,hides,the,sun;
7:  what,time,is,it}
8:  \text{phrase = randomrow(\phrase1)}
9:  \integer{i = items(\phrase)}
10: \statement{Please re-write the following
    sentence in the correct order
11: <center>\embed{reply 1,50x70x\i}</center>
12: }
13: \reply{reply}{\phrase}{type=clickfill}
```

Just copy and paste this source in the same way as the last one, and submit it.

If the e-learning project you want to run contains exercises, Wims can do it better. It's free software, so it can be improved

To answer this exercise, one just needs to click on the blue words, or to drag them to the answer area. The question mark can be used to undo the last word.

As you can see, the primitive "\reply{...}{...}{type=...}" is part of the magic of Wims. According to the type of reply expected, one of the powerful analysers used by Wims will be triggered. Below is a table showing some examples of replies, which are returned if you indicate the right response type.

Tab. 2: What Wims can understand (Wims also deals with fuzzy text, sets and vectors, etc.)

Typed reply	Meaning
2+3/4	2.75 (operations can be performed if the configuration allows it)
2+3/4x	$2+0.75\cdot x$ (symbolic formula can be processed)
R I	The same symbolic value as $R\cdot I$, $I\cdot R$, $R\cdot I^2/I$, etc. It could be about the law of Ohm, $U = R\cdot I$.
1.5e-2 V	0.015 V, the same meaning than 0.015 Wb/s or 0.015 W/A. The underlying engine knows the International System of Units.
1h30min5s	5405 seconds. Hybrid notations are taken in account.
1,2,3 4,5,6 7,8,9	The mathematic matrix with 3 rows and 3 columns (which has a null determinant)
2H2+O2->H2O	The chemical equation $2H_2 + O_2 \rightarrow H_2O$ Wims can check that it is balanced.

The availability of the source code makes it possible to write wrappers that ensure correct collaboration. Adding a new feature to Wims is just a matter of shaping a new glue component, which can be very simple

In order to maintain huge sources of exercises, Wims may generate statements (with true/false replies) on the fly, using powerful randomizers. Here is a list of them.

Tab. 3: Wims has powerful randomizers

Keyword	Meaning
Randchar	Returns a random char taken from a string
Randfile	Returns a random record taken from a text files. Records are multi-line texts.
Randint	Returns a random integer belonging to an interval or a list
Randitem	Returns a random item from comma-separated list
Randline	Returns a random line from a multi-line text
Random	Returns a random floating number belonging to an interval or a list
Randword	Returns a random word from a phrase
Shuffle	Makes a random permutation from a list (options can be specified to choose the parity of the permutation)

In addition, there are fast and powerful tools to combine such randomized data in order to produce coherent exercises, such as evaluators for algebraic expressions, simplifiers, formatters for physical quantities (taking in account a precise number of significant digits), plotters, image generators, etc.

Contribute to the community by creating exercises

Wims enables you to output effective on-line exercises very easily and quickly for yourself and it's worth sharing these exercises with the community. If there are twenty contributors of equal skill in such a community, each of them can author 5% of the contents, and each can benefit from 100% of the product. In addition, this collaboration often increases the quality of the output, as members want to make their contributions to be perfect. Another beneficial side effect is that bugs are more quickly detected and corrected by a group of contributors.

The mailing list for Wims can be subscribed to on the Wims subscribe page (http://www.listes.cri74.org/wws/info/wims) - its information is displayed in French, but many of the messages are in English, so read the archives.

Wims was created by Gang Xiao, who teaches mathematics to first year students at the University of Nice (France). As a consequence the most developed exercise pool is for mathematics, however you can now find an increasing number of exercises in "hard" scientific fields (physics and chemistry, etc). Many of the exercises can be used for other fields and most exercises authored using other tools can be translated to the OEF language.

A subset of Wims can be made compatible with widely accepted standards like SCORM, still a work in progress. However Wims has a unique possibility, which obeys another standard: making software open to human understanding and using open source and free licenses. Other compatibilities have been tested, however at small scale: mathematic exercises not referring to graphics are accessible to blind people, provided they are taught to understand the TEX notation, which is used for any ALT attribute when algebraic formulas are displayed as images. This is also an ongoing work.

Link a Wims server to your educational Content Management System

Gang Xiao, the author of Wims, has created the possibility to link his server to any other CMS, by means of one particular module, which you can find under the Protocol for WIMS direct connection with other web servers (http://wims.unice.fr/wims/?module=adm/raw&job=help).

This kind of link has already been implemented between two Wims servers: the "Campus Virtuel" of Université du Littoral (France) and the Wims server of IUFM de Lille (same country). See the Epistemon project (http://adullact.net/projects/epistemon), which is the CMS developed by Jean-Marie Ball. Developing links with other e-learning systems is a matter of one day of development for a skilled person. You can also find a comprehensive list of free e-learning platforms on the Wikipedia web site (http://en.wikipedia.org/wiki/E-learning).

Install a Wims server in your school

Why install a Wims server, when you can just access other ones on-line?

The reasons for this include: increased responsiveness (the transactions are made on a faster bus), independence from other events (for example the server of a university may be less responsive to external solicitations when students are having exams inside), and hosting custom modules (making extra modules searchable or publicly visible requires the acceptance of the web master).

I know of two methods for quickly installing a Wims server in your school, typically within half an hour:

- Knowims (http://wims.unice.fr/knowims/)
- Freeduc-CD (http://ofset.org/freeduc-cd/)

Knowims is a CD-ROM based on the Knoppix distribution, and customized by Gang Xiao to embed a Wims server. It also contains many applications, which are interesting to maths teachers. Freeduc-CD is also a CD-ROM based on Knoppix, improved on by the OFSET association. It is targeted at end users in elementary and higher schools. It features over 40 interesting educational applications.

Both CD-ROMs will boot on a variety of Intel-based computers, and a Wims server will immediately launch, so you can try it, from the same machine, or from any machine on the same local network (just browse Wims (http://ip.of.the.server/wims/) note the final "/"). You will find instruction on how to train people, discovering Wims, who want to learn how to author simple exercises, create virtual classes, etc.

As long as the architecture of your machine is recognized, it should work, and you should have a proof that the installation on a hard disk succeeded (problems may occur, like power outages, etc, so backup your data). Freeduc-CD comes with many predefined behaviours, which make it easier to launch its services and to add more educational software to those already installed. Currently a CD-ROM based on Knoppix takes up 2GB, but there's more than 2GB of interesting educational free software, and this is still growing.

Another simple method is to use software in binary packages, and install them on a pre-configured server. Gang Xiao offers RPM packages on the Wims site (http://wims.unice.fr) and the OFSET association offers Debian packages (add the line "deb ftp://developer.ofset.org/sarge main" to your sources.list file).

Conclusion

So if the e-learning project you want to run contains exercises, Wims can do it better. It's free software, so it can be improved. If the graphical interface is not as nice as the interface of your preferred web site, please consider contributing to a Cascading Style Sheet (CSS), Wims already supports them. If you dream about a feature you never saw implemented, please contact the author of the article, so we can discuss its feasibility, the glue engine of Wims is not that complicated.

Now, if you want to impress your friends, invite them on a tour of a Wims server. Just use its embedded search engine and type one of the following example keywords (Google won't work, Wims is a web site with an infinite depth, so it blocks web spiders).

- shot
- country
- figures
- animated
- polyray
- vision

Have fun!

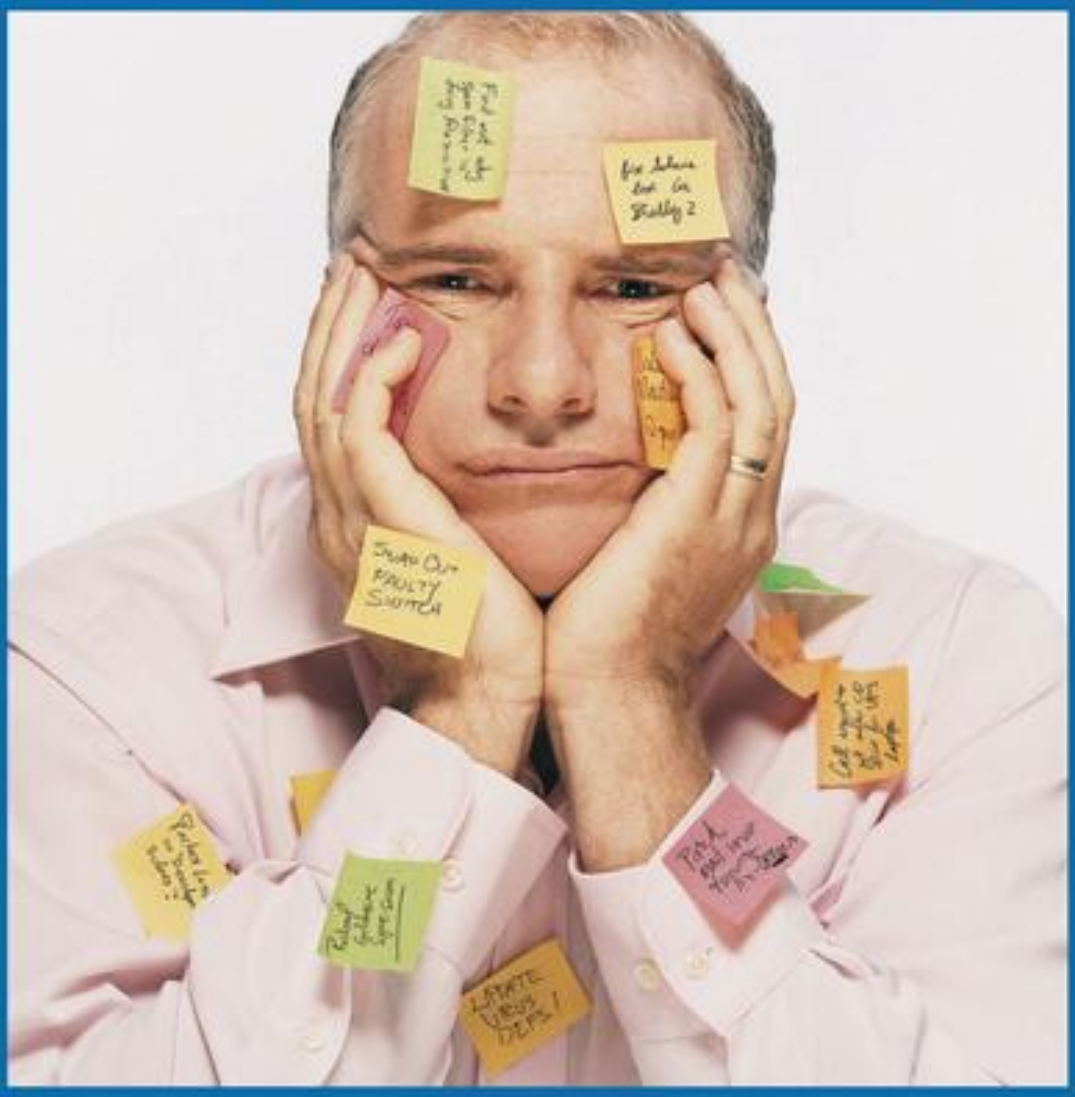
Copyright information

© 2005 by Georges Khaznadar
Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is available at http: //www.gnu.org/copyleft/fdl.html

About the author

Georges Khaznadar is a teacher of physics and chemistry in lycée Jean Bart – Dunkerque, France. He is member of association OFSET – Organisation for Free Software in Education and Training.

Losing track of things?



There is a better way.

RT.

Enterprise-grade issue tracking.
Open source customizability.
Commercial support.

Download it now!

Best Practical Solutions offers training, support and custom development services. Check out our website or contact us for details.

www.bestpractical.com
sales@bestpractical.com

BEST PRACTICAL ™

Worst case scenario - protecting your computer

How to keep sensitive information safe

John Locke

In my last article my laptop had died a spectacular death from a full cup of coffee. I had to send it into the IBM depot, where they replaced nearly everything but the battery. Including the hard drive.

My files were all properly backed up, and I was even able to retrieve the few files I had worked on that day by connecting the drive to another computer. So when the service depot called and said they wanted to replace the drive, I said go ahead.

Now, from a security point of view, the rule of thumb is to destroy all data on hard drives before passing them on. However, if your computer gets stolen you may not get the opportunity. Let's take a closer look about what you can do, why and how.

If you have financial files that include account numbers, or store passwords on your computer, you definitely want to have protection for them

Who cares if someone gets my hard drive?

You may not care. Many people don't. In this day of identity theft, however, being too cavalier about your data may be foolhardy. While there are plenty of other ways that misanthropes have found to hijack your identity, getting financial details off your computer is one of the easy ways, if they get hold of your hard drive. There are basically three reasons to protect data on computers that could be stolen, in increasing levels of paranoia:

1. Because you might get sued or go out of business if the information falls into the wrong hands
2. To prevent identity fraud or theft
3. To protect your privacy

In my business, I work with a lot of different clients. For some of them, I have signed a confidentiality agreement, agreeing not to reveal any of their internal product or business lines. If my laptop were to be stolen with confidential material on it, I could be held liable. This type of information absolutely must be protected.

If you have financial files that include account numbers, or store passwords on your computer, you definitely want to have protection for them. Any geek with a computer could find this stuff on your hard drive, and if the temptation is great, and their ethics loose, they might put your information to misuse.

Even your non-confidential stuff - email, letters, and spreadsheets - may be enough for someone to impersonate you and get credit in your name, or assume your identity when they commit a crime.

What should I pay attention to?

Okay. Let's not get too alarmist here. There are risks involved with setting foot outdoors. In my house, there can

be risks involved without going outdoors. Worrying about the security of your data should not keep you up at night - if it does, I highly recommend you stop reading right now, unplug your computer, run it over with your car, hack it up with an axe, and move to a teepee in Manitoba. I hear there's plenty of deer running around up there, and with our climate changes, there should be some good farming up there soon.

But if you're determined to stay online, just take a moment to think about the kind of data you have on your computers. The same data I considered in my disaster recovery article, and before that, in my password strategy article. Do you have any data you absolutely don't want to have fall in the wrong hands?

Don't bother with email - it has already gone unencrypted through that filthy, spy-infested internet. But do pay attention to your financial records, and especially to any files you've copied (securely, I hope!) from any company file share. If you're responsible for keeping any of that secret, you'd better not leave it unencrypted on a laptop hard drive, especially not in public places.

A general rule of thumb is that increasing security directly hampers convenience. On certain systems, however, encryption has been made very easy to do

For all of the employees out there carrying laptops owned by your employer, you can relax - it's the job of your IT department to make sure their data is properly secured, not yours. But if you have client data, you could be held responsible if it falls into a competitor's hands.

Encryption to the rescue

Luckily, there are some very secure ways to protect your data, using one of a few different types of encryption. I'm not going to get into detail about how encryption works, or what varieties are out there. But I am going to look at three different systems that can be used to encrypt data on your hard drive. They vary based on who can decrypt the data, where you can apply the different encryption types, and how automatic the whole process is.

A general rule of thumb is that increasing security directly hampers convenience. On certain systems, however, encryption has been made very easy to do.

Windows Encrypted File System

This is one area where Microsoft gets it right, with their "Encrypted File System," or EFS. EFS comes with Windows XP Professional, but not XP Home. If you have XP Pro, and your hard drive is in NTFS format, you can encrypt any file or directory by following these steps:

1. In Windows Explorer, right-click the file or directory, and choose *Properties*.
2. Click the *Advanced* button.
3. Check the *Encrypt* checkbox, and click OK.

That's it. Whatever you have encrypted, is now completely secure, even if your hard drive is stolen - unless the attacker guesses your password. EFS works by using strong encryption to hide the data, and then it uses a certificate associated with your login to protect the key. If you log in using another user account, or try to read the files from Linux, you won't be able to get to them.

The downside is, if your administrator resets your password, you lose all access to the encrypted files because the certificate is deleted. It's possible to create a recovery disk *before* you reset your password, but otherwise you're hosed. Another drawback is that you can't back up an EFS file or directory while it's encrypted.

EFS works well for laptops, and I encourage you to turn it on for specific directories, to keep anything you store there safe should you lose control of your hard drive. This system depends upon having a strong log-in password, though, and disabling automatic logins.

Mandrake DrakLoop

Encrypting files, and entire hard drive partitions, is built into most modern Linux distributions. Mandrake provides a nice graphical utility for creating an encrypted drive, called DrakLoop. If it's installed, you can find it under System -> Archiving -> Other. If it's not there, go to the Mandrake Control Center to install software, and search for a package named "mountloop". You'll probably be asked to choose

Fig. 1: Finding DrakLoop in the Mandrake 10.1 menu

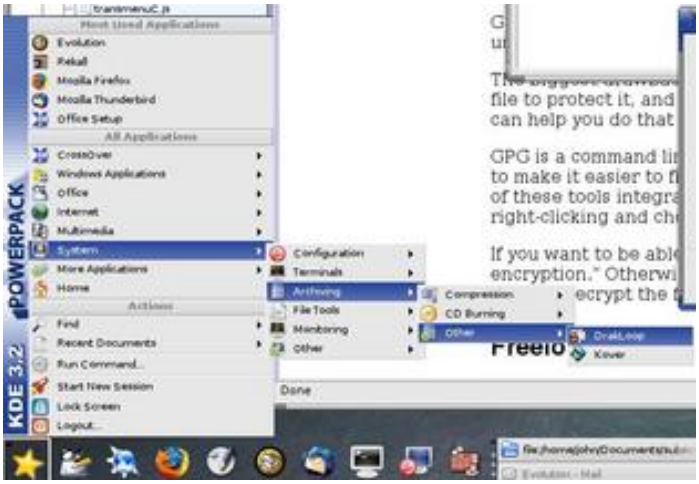


Fig. 2: Setting up an encrypted directory in Mandrake Linux

between a couple of different packages for asking for your passphrase - I've found the "openssh-askpass-gnome" package to be slightly nicer to use.

With DrakLoop, you create a big file of a fixed size, and it is mounted on your system very much like a disk drive. When it's mounted, you can use it like any other directory, storing files, running programs, or whatever you want from this file. When it's unmounted, it's just a single encrypted file, and nobody can determine its contents.

The entire contents of the file is encrypted using a passphrase that you provide. Anybody with the passphrase can decrypt the file and get to its contents. Without the passphrase, it's protected. One advantage of this system is that it's super easy to use, and can be safely backed up in encrypted form with little extra effort. A disadvantage is that you always have to type in the passphrase to mount the file. Another disadvantage is that if you set up additional encrypted directories, you have to type in the passphrase for each one.

To set up an encrypted directory with DrakLoop:

1. Click the Mandrake star, point to **System**, point to **Archiving**, point to **Other**, and click **DrakLoop**.
2. Click the **Add** button.
3. For Directory, type a new path. If you use an existing one, it may delete data in there. For example, I used **Documents/encrypted** for mine.
4. For Size, make it big enough to contain as much data as you'd like to encrypt. If you plan to back up to CD, and have enough disk space, you could make it around 650 MB to make this simple.
5. You can choose between different levels of encryption, from aes128 to aes256. The larger numbers provide even more secure encryption, at the expense of more processing involved. I've stuck with the minimum aes128 for mine.
6. Finally, type in a good strong passphrase in both the password and confirmation boxes. Even though it says password, this should be a passphrase of five or six words, at least 20 characters. Try Diceware (`http://diceware.com/`) for a good way to generate a secure, memorable passphrase.

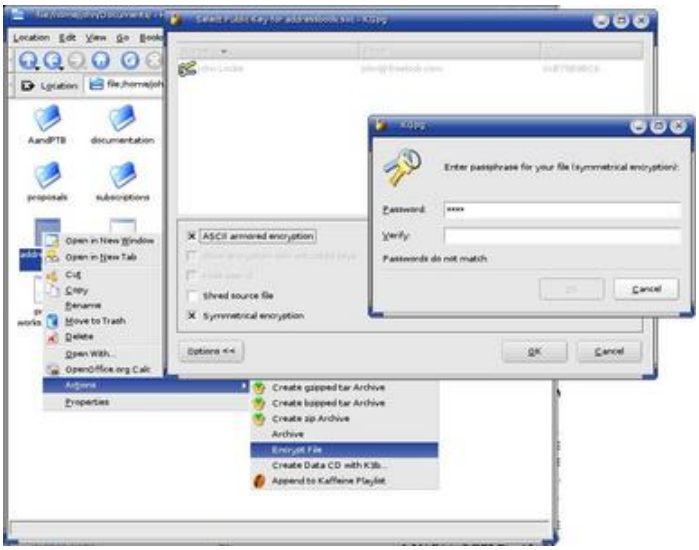
Click **OK**, and you're done!

With DrakLoop, you can always open this program to mount or unmount your encrypted directories. Mandrake will automatically ask for your passphrase when you log in, allowing you to automatically mount the encrypted directories at the start of your session. To encrypt files, simply copy them into the directory you specified, when it's mounted.

KGPG and Windows Privacy Tools

So what if you don't have Windows XP Professional, or Mandrake Linux? The underlying system that DrakLoop uses is available in pretty much any modern Linux system, there just isn't necessarily a nice interface for setting it up. It's also possible to set entire disk partitions to be encrypted, requiring a passphrase to unlock them when you boot the computer. I'll leave it to Google to help you with that.

Fig. 3: Using KGPG to encrypt a file. In Konqueror, right-click the file, and find the encryption option. The figure shows Symmetrical encryption, which uses a passphrase for encryption, instead of a key



But an entirely different way of encrypting files is worth mentioning here. A system called "GNU Privacy Guard", or GPG, provides a way for you to encrypt any file on any operating system. You can encrypt it in such a way that anyone with the passphrase can read it, or so that only specific people can decrypt it. It's based on an earlier system called "Pretty Good Privacy", which infringed on some patent rights and got pulled from the market.

"GNU Privacy Guard", or GPG, provides a way for you to encrypt any file on any operating system. You can encrypt it in such a way that anyone with the passphrase can read it, or so that only specific people can decrypt it

GPG is available for every operating system in wide use, and it's completely cross-platform. The system is used to encrypt email as well as files, and provides several different types of encryption.

The biggest drawback to GPG is that it takes manual intervention to use - you have to explicitly encrypt a file to protect it, and decrypt it before you can use it. But if you have data you need to keep secure, GPG can help you do that *and* share the file with other specific people.

GPG is a command line tool, but it is built into several other programs that can provide a graphic interface to make it easier to figure out. For Windows, try Windows Privacy Tools (http://winpt.sourceforge.net/). For Linux, give KGPG a shot - use the software installer with your distribution to install it. Both of these tools integrate right into the file manager, making it so you can encrypt or decrypt files by simply right-clicking and choosing the action in Windows Explorer or Konqueror.

If you want to be able to decrypt the file with a simple passphrase, make sure you choose "Symmetrical encryption". Otherwise you need to select a person to encrypt the file to, and only the person with that key can decrypt the file.

Encryption is easy

I've only scratched the surface of encryption technologies in this article. It certainly gets a lot more complicated than this, and there are many different systems and ways of using them than I've mentioned here. But what I've shown is that you don't have to be a security expert to use encryption technologies to protect sensitive files.

Copyright information

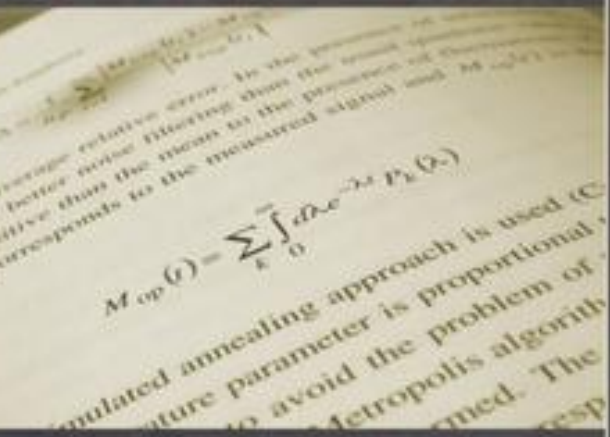
© 2005 by John Locke

This article is made available under the "Attribution-Share-alike" Creative Commons License 2.0 available from http://creativecommons.org/licenses/by-sa/2.0/.

About the author

John Locke is the author of the book *Open Source Solutions for Small Business Problems.* He provides technology strategy and free software implementations for small and growing businesses in the Pacific Northwest through his business, Freelock Computing (http://freelock.com).

Communicating mathematics...



...is getting easier.

Math Focus

News
Discussion
Collaboration
Software
Books

www.mathforge.net

Powerful tools for online mathematics

mathforge

Make it right using Tcl

Software testing with Tcl for Apache Rivet

David N. Welton

Any sufficiently complex software system has bugs, and those of us who aspire to produce high quality work also seek to not only minimize these, but guarantee that our code does what we say it ought to.

One proven way to eliminate bugs, and ensure that code behaves as documented is to test the program. Easy enough to do by hand, when there isn't much functionality. However, when the system grows more complex, and there are many possible environmental factors with various permutations, it quickly becomes obvious that we need to automate our testing.

This article aims to provide the reader with a few notions about software testing in general, and then concentrates on a specific example, using the test suite written in Tcl for Apache Rivet, in order to demonstrate a real-world approach to testing a particular program.

Software testing

Testing is often an afterthought, and even for large, complex systems, or expensive, proprietary software, testing is never going to directly generate revenue, or add new features to the program, and so it has to compete for scarce developer time. Even in free software work, more often than not, it's more fun to spend time hacking on a cool new feature rather than writing test cases to make sure everything works exactly as advertised.

This means that it's important to try to get the most from the time you dedicate to testing - to get better value for money. In technical terms, it is desirable to maximize is the "code coverage" for the time invested. Coverage refers to how much of the code is exercised by the test suite. Think of running a simple program with options -x, -y, and -z. If you run it with the -x option, the "paths" through the other options will not be taken, and you won't know if they really work or not. So you have to expand your code coverage to all three code paths. Generalizing, the two main approaches are "white box" and "black box" testing.

Testing is often an afterthought, and even for large, complex systems, or expensive, proprietary software, testing is never going to directly generate revenue, or add new features to the program, and so it has to compete for scarce developer time

White box testing is testing a program based on knowledge of its internal workings. For example, writing a series of tests that give different input to each C function in a program, and the checking to ensure that it behaves correctly. Obviously, you need the source code, and the ability to rebuild the program in order to do this type of testing. One of the most important reasons to use this approach is that it is theoretically possible to test most or all of the code paths

in the program. In practice though, the effort required to do this may be significant - imagine that you have a C function that takes a struct as input, and that struct is in turn generated by other functions, and so on. It's clear that things can quickly get more comlpicated.

Black box (or "functional") testing involves running a program against a specification to make sure it produces the correct output. For instance, testing to ensure the `ls` program works correctly. A very simple test would be to create a directory with files of known sizes and creation times, run the `ls` program, and compare its output with the known contents of the directory.

Apache Rivet

Apache Rivet is a server-side Tcl system for the creation of dynamic web pages. Think JSP or PHP, but using Tcl, a free, multi platform, general-purpose scripting language. For example:

```
<b><? puts "The date is: [clock format \
  [clock seconds]]" ?></b>
```

It is best to test the software with as little modification to the environment as possible, meaning that the test suite will run using the copy of Apache already installed on the computer - having to create a special copy of Apache would defeat the purpose.

The goal of the test suite is to be able to start the web server with configuration options of our choosing, send HTTP requests, receive answers, stop the server, and then create a report. Because it's so tightly integrated with Apache and Tcl, white box or unit test would be difficult. It would be very laborious to create and fill in all of the arguments that are passed to each function in the C code, because they usually reference complex C structures such as the Tcl interp struct, or the Apache request struct, which rely, in turn, on lots of configuration and set up. The effort required to make most of this work would probably be more than that involved in creating Rivet itself! So, a "black box" testing style would provide more coverage for the time dedicated to it. From the test suite, there much required, as the real work is in devising clever ways to test as much of Rivet's functionality as possible. An application is required that allows program tests to be performed quickly and flexibly, and provides a lot of tools for interacting with Rivet and Apache:

- Reading and writing files in order to manipulate Rivet's configuration files.
- Process control, to control the Apache process itself.
- Sockets and an implementation of the HTTP protocol in order to send requests to the Rivet-enabled web server.
- Good string matching and regular expression support.

Being a fan of Tcl, I choose.

The Tcl Test Suite ships as part of the core Tcl distribution, and is an excellent base upon which to build a series of tests for all kinds of applications

The Tcl test suite

The Tcl Test Suite ships as part of the core Tcl distribution, and is an excellent base upon which to build a series of tests for all kinds of applications. As noted Tcl expert Cameron Laird says:

> Tcl tests automobile engines, emergency telephone circuits, chemical sensors, microprocessors, rocket components, industrial ovens, and much, much more. In the absence of any other knowledge, ANY software project should think of Tcl as its first choice for testing.

What does the suite itself provide? It defines several Tcl commands that are used in order to create and run a series of tests written in the Tcl language, and generate reports based on the results. It gives control over exactly which tests are wanted, and how the output is to be organized, as well as a number of commands to control the testing environment (files, directories, output, errors, and so on).

Returning to the example of testing `ls`, the following code contains some simplistic tests for the `ls` command.

```
# Load the tcltest package, and import its commands
# into the current namespace.

package require tcltest
namespace import tcltest::*
```

```
# Set up a directory named 'testdir' for the tests
# to use.
tcltest::makeDirectory testdir

# ls -l output regexp for two files that will
# be created.
set lslformat "^-rw-rw-r-- +1 $tcl_platform(user) \
        +\\w+ +1 \(.+\) A$ ^-rw-rw-r-- +1 \
        $tcl_platform(user) +\\w+ +1 \(.+\) B$"
# perms username group time filename

# Create two empty files.
proc makefiles {} {
  makeFile {} testdir/A
  makeFile {} testdir/B
}

# Delete the same two files.
proc delfiles {} {
  removeFile testdir/A
  removeFile testdir/B
}

# Running ls on an empty directory should return
# an empty string result.
test ls-1.1 {empty dir ls test} {
  exec ls testdir
} {}

# Run ls on two files.
test ls-2.1 {ls two files} {
  makefiles
  exec ls testdir
} {A
B}

# Delete the files for the next run.
delfiles

# Run ls -l on two files. Matching on this is more
# complex because of the data and user. The username
# from comes from $tcl_platform(user). To match the
# date, save the time the files were created, and
# check that against the time reported.

test ls-3.1 {ls -l test} {
    makefiles
    # Save the file creation time.
    set filecreatetime [clock seconds]

    # Run ls -l
    set result [exec ls -l testdir]

    # Check the results against the regular expression,
    # and add that to the result.
    lappend testresult \
        [regexp -line $lslformat $result \
             match matchtime]

    # If the regexp matched, have a look at the time
    # it returned, and compare it with the time we
    # have in memory.
    if { $testresult == 1 } {
        # clock scan reads the formatted time string
        # and returns a number of seconds.
        set lsfiletime [clock scan $matchtime]
        # Make sure the numbers aren't too different.
        # A small difference is ok here, because the
        # 'clock' command might have been run a little
        # later than when the files that were created,
        # and more importantly, because ls -l does not
        # include seconds in its results.
        lappend testresult \
          [expr {abs($filecreatetime - $lsfiletime) \
                 > 100}]
    }
    set testresult
} {1 0}

delfiles

# Clean up tests and print results.

cleanupTests
```

In this script, the test cases are defined by the "test" command. In this code fragment, for instance

```
test ls-2.1 {ls two files} {
        makefiles
        set result [exec ls testdir]
        set result
        puts $result
} {
A
B
}
```

`ls-2.1` is the short name of the test, which is followed by a description of the test, the test case body, and finally, the expected result.

The body of the code in this example does nothing more than create the two files and run `ls`. The result should be A B, each one on its own line. When the test is run, it should show that it passed all of the tests:

```
@ashland [~] $ tclsh ./lstests.test
lstests.test:
Total 3
Passed 3
Skipped 0
Failed 0
```

Tcl has lots of facilities for interacting with the outside world, so you can open sockets, look at files, execute programs and so on

If there had been a failure, it might look something like this:

```
@ashland [~] $ tclsh ./lstests.test

==== ls-2.1 ls two files FAILED
==== Contents of test case:

        makefiles
        exec ls testdir

---- Result was:
A
B
C
---- Result should have been (exact matching):
A
B
==== ls-2.1 FAILED

lstests.test: Total 3    Passed 2    Skipped 0    Failed 1
```

In this case, there is a third file (for the sake of argument), and so the test fails. The test suite shows which test failed, what the expected results were, and how many of the tests passed. It is also clear that in the 3rd test, the "correct output" is not just the results of the `ls -l` command. It can be calculated whether the `ls -l` output is satisfactory by both testing it against the regular expression, and examining the time. So instead of a basic string match, return the results of the "real" matching performed in the test body and "match" against those.

Because this framework is so simple, it can be adopted to any number of situations. All that is required is find a way to make the test perform an action, and then compare that result with the expected outcome. Tcl has lots of facilities for interacting with the outside world, so you can open sockets, look at files, execute programs and so on.

Testing Apache Rivet

In order to test Apache Rivet, it's necessary to be able to create a controlled environment in which to run the tests, then run through them, stopping and starting the server as the tests dictate. An Apache module Rivet can be either: compiled directly into the web server, or, of course, loaded at run time when compiled as a shared object. In either case, it is just one component of the web server, which is a complex system. For this reason, it is necessary to ensure that testing it gives the same results, even across diverse systems, so that programmers can count on it to always execute their code correctly.

The first task is to set up a minimalist environment for Rivet to run in by creating configuration files with as little in them as possible. Since the aim is to automate testing, advantage is taken of several Apache features in order to automatically generate these configuration files.

It's necessary to be able to create a controlled environment in which to run the tests, then run through them, stopping and starting the server as the tests dictate

```
@ashland [~/workshop/tcllib/modules/log] $ apache -V
Server version: Apache/1.3.26 (Unix) Debian GNU/Linux
Server built: Aug 12 2002 21:19:30
Server's Module Magic Number: 19990320:13
Server compiled with....
-D EAPI
-D HAVE_MMAP
       -D HAVE_SHMGET
       -D USE_SHMGET_SCOREBOARD
-D USE_MMAP_FILES
-D HAVE_FCNTL_SERIALIZED_ACCEPT
-D HAVE_SYSVSEM_SERIALIZED_ACCEPT
-D SINGLE_LISTEN_UNSERIALIZED_ACCEPT
       -D HTTPD_ROOT="/usr"
-D SUEXEC_BIN="/usr/lib/apache/suexec"
       -D DEFAULT_PIDLOG="/var/run/apache.pid"
    -D DEFAULT_SCOREBOARD="/var/run/apache.scoreboard"
       -D DEFAULT_LOCKFILE="/var/run/apache.lock"
       -D DEFAULT_ERRORLOG="/var/log/apache/error.log"
       -D TYPES_CONFIG_FILE="/etc/mime.types"
       -D SERVER_CONFIG_FILE="/etc/apache/httpd.conf"
       -D ACCESS_CONFIG_FILE="/etc/apache/access.conf"
       -D RESOURCE_CONFIG_FILE="/etc/apache/srm.conf"
```

The command `apache -V` gives a variety of information about Apache's compile time options, which can then be put to use...

```
@ashland [~/workshop/tcllib/modules/log] $ apache -l
Compiled-in modules:
        http_core.c
        mod_so.c
        mod_macro.c
suexec: enabled; valid wrapper /usr/lib/apache/suexec
```

`apache -l` reports the modules that are already compiled into the Apache build. On my Debian system, very little is actually compiled in, as you can see - it's all loaded as modules, in order to make the Apache install as flexible as possible.

Thanks to this information, it's possible to figure out if Rivet has been compiled into the server, and where the server's main configuration file is. In this case, it can be seen that mod_so is present, meaning modules can be loaded, Rivet is not built into the executable, and that the main configuration file is at `/etc/apache/httpd.conf`. Were Rivet compiled directly into the main executable, a line like "mod_rivet.c" would have been seen, and the next step could be taken. If there were no mod_rivet, and no mod_so to load other modules, it would have been necessary to error out! The Tcl code for retrieving this information - apachetest::getcompiledin - isn't very complex, and is really just there to sort through the output of "apache -l". Since there is a way to load shared objects, take a look at what happens next.

The `getcompiledin` procedure tells which modules are part of the apache build, and that Rivet isn't one of them. Given that the standard use of the test suite is to test a change that has just been made to the Rivet source code, this situation is actually sensible, because it's easier to quickly compile Rivet as a module, and then include that, than make a big, fat build of Apache. In fact, given the Rivet directory layout:

```
tcl-rivet/
src/
tests/
```

It's easy to find the Rivet module that was just built – the test suite is always run in the "tests" directory, so the new shared object just built is only "one directory over" or in Unix parlance, "../src".

When writing a test suite for your own code, this sort of arrangement is helpful, so that you don't have to "tell" (via command line options or some other means) your test suite where everything is, making it even faster to run. Getting back on track, what's built in and where Rivet is are known, but to run all the tests we want to perform, a few other Apache modules are needed: mod_log_config, mod_mime, mod_negotiation, mod_dir mod_access, and mod_auth, which are all part of the standard Apache distribution. Even if they aren't compiled in, no problem, in apachetest::getloadmodules, sift through the configuration file, looking for the locations of those modules that are needed, or error out of they're not there, because they really are required!

Now that it's known where everything required is, a minimalistic configuration file can be written in the tests/ directory. I'll come back to this file and its contents in a moment. You should now be able to run Apache with Rivet loaded up, which is a pretty good place to start the tests! Here, another obstacle is encountered though. Apache is a "daemon" (i.e., on Unix, a program that runs in the background) and you want to be able to start and stop Apache at will, so it can't just go "running loose". Fortunately, once again there is a command line option that comes in quite handy: "-X". When apache is run in this way, it doesn't go into the background, but instead runs as one process in the foreground, which is a big help. Apache still needs to be launched in such a way that the program doesn't block (stop and wait) while it is running, because then you couldn't go about testing. Tcl's exec command permits this with no problems:

```
set serverpid [eval exec $binname -X -f \
            [file join [pwd] server.conf] \
            $options >& apachelog.txt & ]
```

What is being done here is the apache server process is being executed in the background (via the & at the end of the command line), and with the -f command line option, telling it to use the configuration file just created, instead of the standard one. The process id must be monitored, so that it can killed off when required, and any output redirected (there shouldn't be much) to a file. The configuration file created is very simple, just enough to make Rivet and a few other modules, that it interacts with, work. Rivet has several configuration directives that it is desirable to test, of course, and furthermore, you need to be able to modify several other lines in the standard configuration that is used. In order to be flexible, utilize a template file, called, `template.conf.tcl`, which has several variables that are filled in when loading up the file. After which, the file is then written to the tests/ directory, where it is ready to be used.

```
# Minimal config file for testing

# Parsed by makeconf.tcl

ServerRoot "$CWD"

PidFile "$CWD/httpd.pid"

ResourceConfig "$CWD/srm.conf"
AccessConfig "$CWD/access.conf"
```

```
Timeout 300

MaxRequestsPerChild 0

$LOADMODULES

Port 8081

ServerName localhost

DocumentRoot "$CWD"

<Directory "$CWD">
Options All MultiViews
AllowOverride All
Order allow,deny
Allow from all
</Directory>

<IfModule mod_dir.c>
DirectoryIndex index.html
</IfModule>

AccessFileName .htaccess

HostnameLookups Off

ErrorLog $CWD/error_log

LogLevel debug

LogFormat "%h %l %u %t \\"%r\\" \
           %>s %b \\"%{Referer}i\\" \
           \\"%{User-Agent}i\\"" combined
CustomLog "$CWD/access_log" combined
```

After all of this, it's evident why it was desirable to automate it, eh? By way of explanation, you can see that the file is littered with `$CWD`, which you should replace with the current working directory where the tests are located. All of the test and .rvt files are located in one big directory (perhaps it's time to sort things out a bit more!), so that locating them is no problem - set `ServerRoot` and `DocumentRoot` to CWD. `$LOADMODULES`, which is replaced by directives to load the modules required - the same ones Apache regularly uses. Aside from the configuration files, .test and .rvt files, there are a few log files, but (currently) nothing is required of those. There is no other preparation required for our environment. For Rivet, there is no cleanup to perform either, but you should always consider this, in order to have a pristine condition to start from, after your tests have run.

Finally, the first test is ready to be run. Of course, tradition calls for "Hello, world" at this point, and who are we to buck the trend? The `hello.rvt` file looks like this:

```
<?
# hello-1.1
puts ''Hello, World\n''
# i18n-1.1
puts ''< À È Ì Ò Ù - El Burro Sabe Más
Que Tú!\n''
?>
<p>ÆüËÜ¸ì(EUC-JP Japanese text)</p>
```

Leaving be the Spanish and Japanese texts (used to test internationalization features) for the moment, take a look at what happens in hello.test:

```
set testfilename1 hello.rvt

test hello-1.1 {hello world test} {
 set page [::http::geturl \
  "${urlbase}$testfilename1"]
 regexp -line {^Hello, World$} \
  [::http::data $page] match
 set match
} {Hello, World}
```

Which is really pretty simple.
First, the URL with Tcl's http package is fetched:

```
set page [::http::geturl "${urlbase}$testfilename1"]
```

The page data is run through a regular expression, and put the results in the "match" variable...

```
regexp -line {^Hello, World$} [::http::data $page] match
```

... which is then returned.

```
 set match
} {Hello, World}
```

The result of this test should be "Hello, World". If the "match" variable doesn't contain that text, the test will fail. Most of the tests are similar to this one. HTTP isn't a terribly complicated protocol, so for most cases, all that is necessary is to do is send a request, and check what is received back from the server. An effort has been made to test as many different features of Rivet as possible, including:

- The ability to handle internationalized content - at the moment, a few accented characters are sent, and some Japanese text.

- Binary data, which is tested by sending and receiving a JPEG of Fishbone's Angelo Moore! This required the use of some code to do a file upload in Tcl, which I gratefully borrowed from Jeff Hobbs. To make sure the result is what it is supposed to be, compare the original file with the uploaded or received version.
- Error messages. Just to make sure that the error message works correctly.
- Apache's built-in environmental variables like (like `DOCUMENT_ROOT`).
- Passing variables through both GET and POST.
- Cookies. These two were very easy, thanks to Tcl's http package.
- Parsing and including secondary files, with and without non-ASCII characters. Rivet is able to "include" other Rivet (mixed HTML and Tcl) files, parsing them correctly as it does.
- Internationalization in general. This was a difficult one - despite speaking Italian, I'm a native English speaker and am not too concerned about creating content with non-ASCII character sets. The value of the free software community was proven when two Japanese users, Taguchi Takeshi and Makoto Satoh provided me with help in understanding the issues involved, and how Rivet needed to behave to meet their needs.
- Uploading files.

Getting Apache's configuration directives right in Rivet was also somewhat of a chore, and it was necessary to ensure that they weren't interacting badly with one another. To cover all the possible permutations, the creation of tests, looping through and trying different combinations, was automated:

```
foreach Config $ConfigList {
 incr i
 test config-auto-${i}.1 {config test} {
  ... body ...
 }
}
```

`i` is a counter variable, that is used in creating the test name: `config-auto-${i}.1`, and incremented `(incr i)` every time around.

HTTP isn't a terribly complicated protocol, so for most cases, all that is necessary is to do is send a request, and check what is received back from the server

Conclusion

Hopefully, this article has stimulated some interest in a subject that at first glance doesn't appear to be all that exciting. It still may not be as fun or glorious as adding the greatest new kernel feature, but can be an entertaining challenge to try to devise the best ways to test your program's features one at a time. Of course, the benefits to your code quality will speak for themselves.

Copyright information

© 2005 by David N. Welton
This article is made available under the "Attribution-Share-alike" Creative Commons License 2.0 available from http://creativecommons.org/licenses/by-sa/2.0/.

About the author

David N. Welton lives and works in Padova, Italy as a consultant specializing in free software technologies (Linux, Apache, Tcl, Python, C, etc...) providing programming, training and strategic consulting services. He is an active participant in the Tcl comunity, has been a part of the Debian project since 1997, and has been the Vice President (cat herder?) of Apache Tcl with the Apache Software Foundation since 2001. He may be found on the web at the DedaSys web site (http://www.dedasys.com/davidw/)

Center of the System Administration

www.sbin.org

Open look at common things

Go Open

Open Source Software, which originally developed out of the passion of hackers and geeks, has now become the basis of real business solutions. Low TCO, high reliability, security and performance of Open Source software make the shift from "closed" technologies and standards to modern, open solutions inevitable.

CSA offers development, deployment and support services based on Open Source technologies and standards.
Up-to-date solutions for small and medium size businesses are available for you today with guaranteed full 24/7 support and no license fees.

Today, being closed means being left behind

+7 8793 365584

Designed by ITA "InfoMaster"

net.labels

An introduction to the emerging phenomenon of net.labels

Adam Hyde

The traditional approach to releasing music, independent from what is called "the music industry", follows a basic formula: record, print CD, promote, distribute, promote, lose money.

It is difficult to know why so many independent musicians follow this pattern, seemingly oblivious to the fact that they will almost certainly lose time and money. Thankfully there's a new type of independent music label that is emerging on the internet. These entities call themselves *net.labels* and are in the process of defining an interesting new subculture of independent music distribution.

There's a new type of independent music label that is emerging on the internet. These entities call themselves *net.labels* and are in the process of defining an interesting new subculture of independent music distribution

Sharing music

It needs to be said that sharing music online has been going on for a very long time. Perhaps the first substantial example of this was with the sharing of *Tracker* files over *Bulletin Boards*. *Trackers* are a type of sound software that use *mod* files for storing instructions. These files are then re-interpreted by other *Tracker* software for replay or for the purpose of creating a derivative work (the *mod* files are re-editable through the *Tracker* software).

Fig. 1: net.label noiseusse

I am going to skip this very interesting and important phase of online audio and file sharing and look at the more recent history leading to the development of *net.labels*. This story starts largely at the moment the music industry decided to introduce *digital audio* to the consumer market.

Before the music industry invented a way to resell us the same music we already owned there were not many complaints about the quality and function of vinyl. In fact when the new media was introduced there was a lot of grumbling, especially by music aficionados because CD audio at that time was an unknown proposition. The physics of this new media made it sound like the technology was in-

herently inferior and more expensive. Breaking up analog audio into a series of ones and zeros just sounded like a bad idea. How could a mathematical representation of an analog wave (sound wave) possibly sound as good, or, as the CD advocates claimed, *better* than an analog representation of an analog wave? It just didn't sound right (excuse the poor pun).

However, the music industry led the charge, and good for them, as there are now few complaints about digital audio. Few complaints... apart from ones that come from those in the music industry. There is of course some irony at work here: the music industry (the people that made us buy all of the music we already owned in the new format - digital audio) are now enraged when consumers ingeniously utilize two of the inherent advantages of the new medium for their own gain: the ability to copy content quickly, and the ability to distribute content virally. Digital audio allows an ease of replication unmatched by any analog audio format that has ever existed.

File sharing

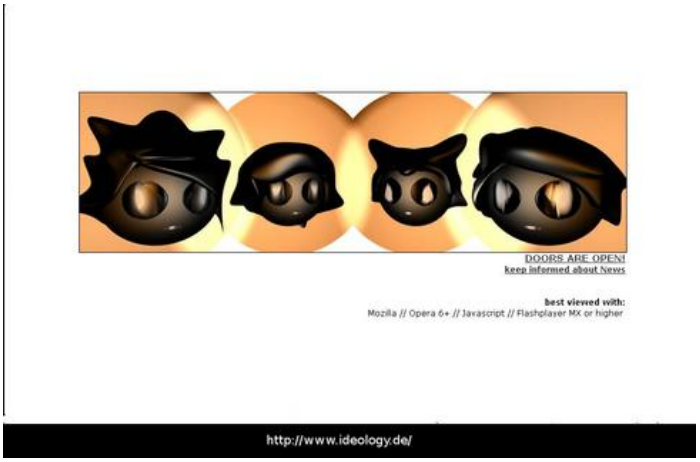
In the "underground" of *News Groups* file sharing had been going on for some time, largely unharnessed by the surface web dwellers. This was more or less ignored by the industry because the logistics of this kind of distribution meant that only a small minority of geeks participated. Files were generally split into parts and compressed. To put them back together again all of the parts had to be found, de-compressed, and re-assembled. And if one part was missing too bad.

This subculture also didn't suffer fools, so if you hadn't been born with the innate knowledge of how to compress and de-compress *rar* files then there was no point asking, you were by definition, the wrong species to try it in the first place.

Audio file sharing couldn't get moving until easy to use consumer technology came along. This technology was of course, peer-to-peer (*P2P*) file sharing as demonstrated most prominently by *Shaun Fannings'* dorm room invention - *Napster.*

So the file sharing industry was born. Interestingly for independent music producers, this new technology, combined with *MP3*, also enabled the distribution of independent music at a fraction of the cost of traditional methods.

Fig. 2: net.label ideology


DOORS ARE OPEN!
keep informed about News
best viewed with:
Mozilla // Opera 6+ // Javascript // Flashplayer MX or higher
http://www.ideology.de/

P2P labels

As a starting point for examining this emerging phenomenon take a look at a *net.label*, which uses *P2P* networks for distributing its content. Most *net.labels* do not, in fact, utilize this method as their primary distribution technology, preferring to allow users to download the music direct from their homepages, with *P2P* as a secondary strategy for distribution. For starters take SoulSeek Records (http://www.slsknet.org/).

S*oulSeek* has a special place in my heart, as it was one of the file sharing sites I used most before the music industry became the bogeyman and closed most successful file sharing networks down in a legislative rampage. *SoulSeek*, instead of dying, transformed itself into a new model of record label.

Although you can use *SoulSeek* to share music illegally, it tries to distance itself from this practice and indemnify itself as its policy now reads:

> "Soulseek(tm) does not endorse nor condone the sharing of copyrighted materials. You should only share and download files, which you are legally allowed to or have otherwise received permission to share. Soulseek(tm). was created to encourage the sharing of public domain music from unsigned, independent artists willing to share their work and communicate with a large audience in an efficient way." (SoulSeek Records rules) (http://www.slsknet.org/)

Fig. 3: net.label Kallisti Gold

However, *SoulSeek* doesn't do much to make concrete its position as an independent music distributer as opposed to a file sharing network where many illegal files get shared. In fact it exists in rather vulnerable territory as the operators of this system charge money to users that want "priority access" (you pay to be first in line) and this is something the *Recording Industry Association of America* (*RIAA*) will likely take a rather dim view of when *SoulSeek* finally appears in their radar. However *not appearing in their radar* seems to be the only defensive tactic of *SoulSeek* as founder *Nir Arbel* said when interviewed by slyck.com in SoulSeek Records news (http://www.slsknet.org/news.php?story=356):

> **Slyck.com:** Considering mostly techno, dance, house and electronic music is on the network, do you feel this has prevented the RIAA from taking an interest in SoulSeek?
>
> **Nir Arbel:** It couldn't have hurt. Considering the majority of stuff trafficked on the system is probably non-RIAA owned, and seeing as the system is pretty small to begin with, it's likely we're not even on the RIAA's radar.

This doesn't look like a smart or sustainable strategy to me. Considering the paranoia of the *RIAA,* I wouldn't bet on *SoulSeek* lasting the distance.

S*oulSeek* doesn't work for me as a *net.label*. It doesn't give the special feeling of trust you get from a label that you can expect to deliver the music you need, instead it feels like a file sharing network - a particularly good *P2P* network but far from the sort of entity we associate with the idea of an independent music label. To me a label is something I can trust, it has a character that I relate to, without this trust I won't spend the time obtaining and listening to material because it might turn out to be awful nonsense. Essentially this is the problem of positing a file sharing network as a new kind of independent music label... the point of the *P2P* technology is that users have the ability to share what they like. As the consumer, you are given no guarantees, you roll the dice, and you take the risks. To try and combat this many operate as pseudo self-regulating social networks, of which *SoulSeek* is a good example. In these situations there are strong rules of sharing etiquette and online behavior. However, they just don't fill the role of a *net.label* well; the content available isn't exclusively independent releases, and it's largely illegally shared copyrighted material. This doesn't mean there can't be an effective strategy for *net.labels* that utilize file sharing networks as their main distribution technology. So far however I haven't found an approach like this that satisfies my needs as a user in search of a trusted source of independent music.

P2*P* doesn't offer much control over the presentation of the content either. If an artist wishes to release an EP where the track order is important then this cannot easily be maintained with these kind of distribution networks. This also applies to the associated information and artwork (it is difficult to maintain the association using *P2P*).

net.labels

A *net.label* like ComfortStand (http://www.comfortstand.com) moves more in the direction of becoming an interesting model for a *net.label* and it fulfills a lot of the necessary requirements for what I think makes a good *net.label*. The site is a year old, well organized, and offers a lot of material in their catalog of artists. Each release is supported by "cover art" and "liner notes" (two terms they use on their site which are derived from the earlier vinyl era). You download an all-in-one compact *zip* file, or each track individually. It's also possible to listen to the content before you 'buy' (download) through *Streaming Flash* or *MP3*. Some releases even come with CD art to print and stick to your freshly burnt CD. The model is very much like a record label which has been put online

with very little or no translation into the new medium, you are expected, in some cases, to complete the process of manufacture and make the CD yourself, complete with cover and disc art.

This is, I think the limitation of this approach; it feels to me too much like it's an extrapolation of an established way of doing things and not a creative exploration of a new way of doing things with new technologies. The label does offer itself as a trusted source, which is largely down to editorial policy. Whether their choices are ones you enjoy is another matter, but I have to say that the straight-forward *Movable-type* blog-like interface deters me from downloading anything from this site.

As a tester I downloaded their latest compilation called *Comfort Cake*, which is a collection of their favorite tracks for the first anniversary celebrations. I notice when downloading the *zip* file (140MB) that the files are hosted on The Internet Archive (http://www.archive.org). *The Internet Archive* is an interesting service that archives the internet by running an automated software process (known as a *spider*) to crawl through the web and download every page that exists on the web. Copyright nightmare as it may be, its an amazing service and it allows anyone to browse previously stored versions of almost any website that has ever existed. In addition to this *The Internet Archive* offers a service to *net.labels* so the labels can host their content free of charge. There are a few other services like this including *scene.org* but none as comprehensive and as fast as *The Internet Archive's* service. For more information see the archive.org net.label service (http://www.archive.org/audio/netlabels.php).

The *Comfort Cake* release was predictably cheesy and goes to prove that you can judge a *net.label* by its homepage.

Refining the idea

Another approach is undertaken by 20kbps (http://20kbps.sofapause.ch/), which is a *net.label* run out of Switzerland. This site is deliberately low-fi, an approach I personally enjoy. I respond to this better than to a site like *ComfortStand* which is nice but a bit too sterile. The aim of the label is to introduce music that explores the aesthetic of low-fi audio, it could be likened to well known labels like New Zealands *Corpus Hermeticum* in that sense. Past the front page is a listing of news about releases and a direct link to the *MP3* files. The music is also stored at the fabulous net.label hosting facility at *The Internet Archive*. The attitude is inherent in the site - if you like one release then you share the same ethic and hence you are part of the *20kbps* clan. The texts take a very direct tone about the releases and you get the feeling that you'll either love or hate what they do.

Fig. 4: net.label 20kbps

The cool thing is that the releases r*eally* are 20kbps and hence are very low-fi, defying the boring and relentless push towards "perfect" digital sound. I like the artifacts created by high compression and these guys do some cool things exploring that territory. It's an aesthetic, a way of doing things, and an attitude. The extra bonus is that because of the high compression used the files are really small and download extremely fast - a four track release (and most net.labels still use terminology like "EP") weighs in at about 4MB! I found their approach so complete that I was persuaded to listen to material beyond the boundaries of what I'd normally enjoy. The point is that I might not always enjoy what they do but I trust them to always do something interesting.

I like this label a lot but it might not be for everybody.

Looking at something a little more hi-fi check out the Lithuanian label Sutemos (http://www.sutemos.net). This is very slickly designed and comes complete with its own web magazine with reviews, interviews and articles about the *Sutemos* artists and releases as well as interviews with other musicians like *Monolake* and *Chris Cunningham*.

This site embraces the new medium a lot better than the very traditional approach of *ComfortStand*, presenting a blog-

Fig. 5: net.label sutemos

Fig. 6: net.label falsch

like interface, throwing away the default styles but with a great design. The result is a site that looks good and (if you have used *blogs*) is intuitive to navigate. Also, instead of cover-art and liner notes we have "virtual galleries" (although I don't like the use of the word "virtual" in any situation) which are photos and images associated with the release. They are not meant to be cover-art but are just there to be looked at and admired either online or after you have opened the *zip* file for the release.

You can also subscribe to their email newsletter if you get hooked. The releases are typical "LP" length format which hopefully more *net.labels* will fight to root out from the norm, there is after all no need to be limited by concepts set as a standard for outdated formats. A 'release' no longer needs to be considered anything more than a snapshot or version of a "track" at a particular moment in time, instead of as something than has been etched forever by laser or lathe and can't be changed, deleted, updated, or rolled back.

The interesting motivation of *Sutemos* is that it seems to see itself as a half-way step toward being a "real" record label. The release notes (released in Microsoft Word format, but that's another battle) for *3tronik* state:

> "Although it is improper in society to boast I will let myself say that *Sutemos* net label is getting rid of its virtual features gradually and is releasing stuff that would almost fit the CD/LP format. The album of French producer *Florin Fabien (aka 3tronik)* called *Mirror* will take another fast step to prove that to you. I think that after listening to the newest work of *3tronik* you will agree that it is more than just an mp3 release and we are sincerely hoping that after *Mirror* he won't need to appear on *Sutemos* (or on any other net label) again. This is what we wish him."

The label is providing a stepping stone opportunity for the artist, allowing him to be noticed and released on a bona-fide, hard format record label. This has some form of altruism attached to it, but also fundamentally it is a statement about the cultural economics of *net.labels*. Each of these enterprises is discovering its own area and its own economy, some appear to exist on an economy of reputation, others like *Sutemos* see themselves as providing artists with a promotional medium from which they can refine their skills, develop a following and step into another economy perhaps where actual cash changes hands.

That's not to say it's impossible to establish a *net.label* that sells music. It is certainly possible but the phenomenon has not yet embraced this strategy to any great degree, perhaps because selling music online also sets a logistical and financial threshold to building an audience. Most *net.labels* seem to be more interested in how many downloads they get rather than how much money they make. Additionally it could be said that *net.labels* are throwing away the overburdening infrastructures required to make money, establishing and managing these structures after-all are the very things that have lost many independent labels money in the past.

Finally, falsch (`http://fals.ch`) is interesting because it takes a completely different tactic from other *net.labels*.

Born from the Austrian *mego* and *farmers manual* borg, this site is fantastic due to its in-your-face design and tricky navigation. This site will almost certainly cause cerebral hemorrhaging for technophobes, offering no shelter for the lost or overwhelmed.

Licensing

The question now arises... isn't downloading music illegal? Well this is the problem underlying the current war on file sharing, there is little done to discriminate between legal and illegal file sharing.

> Audio file sharing couldn't get moving until easy to use consumer technology came along. This technology was of course, peer-to-peer (*P2P*) file sharing

There's a lot of content online that is free and available for download legally. If you want to check out some of the repositories for this kind of content then have a look again at *The Internet Archive* or Legal Torrents (http://www.legaltorrents.com/) or the activist video sharing network v2v (http://v2v.cc/) and there are many others. *Bittorrent*, by the way, is an extremely interesting technology that aggregates available bandwidth from hosts that have the file you want so your download proceeds faster than if it was downloading from just one host. This technology is used for some *net.labels* including the Croatian label egoboobits (http://www.egoboobits.net).

American academic *Laurence Lessig*, inspired by the recent moves on open source software licensing, has made a number of template licenses for distributing content. These are bundled together in a site (movement) called Creative Commons (http://www.creativecommons.org).

These licenses give you the permission to set the parameters of the distribution and re-use of your audio content.

However the scope of these licenses is much broader in philosophy, and central to what they are doing is the idea of stimulating creative development through derivative works. *Laurence Lessig* wants to encourage artists to consider licensing material so that it can not only be distributed but also so that it contributes to a pool of creative works that can be drawn on as raw materials by other artists. His

Fig. 7: net.label Kikapu

http://www.kikapu.com/

claim is that culture has always developed this way and that by unnecessarily restraining this process through strict content licenses we live in a poorer cultural environment. That is a point that you need to decide for yourself, but you may find *Laurence Lessig's* book *Free Culture* an interesting read on these issues. *Free Culture* is available for free from http://www.free-culture.cc/ (http://www.free-culture.cc/)

Conclusion

net.labels are refining technological processes and defining a cultural economy and identity for independent music distribution. Evading the financial burdens that envelop CD production and distribution, these pioneers are cutting an interesting new path which is well worth the download time if you are a fan of independently produced music.

net.label links

12 rec (http://www.12rec.net)
2063 music (http://www.2063music.de)
20kbps (http://20kbps.sofapause.ch)
8 Bit Peoples (http://www.8bitpeoples.com)
Acedia Music (http://www.acediamusic.org)
Binary Zero (http://www.binaryzero.com)
Broque (http://www.broque.de)
Cyclene (http://www.cyclene.com)
falsch (http://fals.ch)

Fukkgod (http://www.fukkgod.org)
Ideology (http://www.ideology.de)
Kahvi Collective (http://kahvi.org)
Kallisti Gold (http://www.kallistigold.com)
Kikapu (http://www.kikapu.com)
Linguablanca (http://linguablanca.sevcom.com)
Maetrix Solution (http://www.maetrixsolution.com)
Monotonic (http://www.monotonik.com)
Noiseusse (http://noiseusse.org)
privatelektro (http://www.privatelektro.de)
pumplanerecords (http://www.pumplanerec.com)
realaudio.ch (http://www.realaudio.ch)
rand (http://r4nd.org)
Repan Records (http://www.repanrecords.se)
Sofa Sound (http://www.sofasound.de)
Sweet Smelling Surfaces (http://www.sumodehouse.com/sss)
stadt gruen (http://www.stadtgruenlabel.net)
statsifield (http://stasisfield.com)
stromklang (http://www.stromklang.de)
Subsource (http://www.subsource.de)
Surfaces (http://surfaces.tinkle.lt)
sutemos (http://www.sutemos.net)
yrnuid (http://www.yrnuid.net)

Copyright information

© 2005 by Adam Hyde
This article is made available under the "Attribution" Creative Commons License 2.0 available from http://creativecommons.org/licenses/by/2.0/.

About the author

Adam is a digital artist (r a d i o q u a l i a (http://www.radioqualia.net/)) who works with software, online audio and video, sound art, new technologies and more traditional forms of broadcast. See also r a d i o q u a l i a's latest project Radio Astronomy (http://www.radio-astronomy.net)

ANSWER SQUAD.com
Tech Questions? We wrote the book!

Feel like roadkill on the Information Superhighway?

Questions about your computer, the Internet, the program you're writing, or even just the commercial application you're stuck working with every day? You've tried searching the Internet for solutions, with mixed results. You've asked people on various mailing lists, just to be flamed or answered with "rtfm!" You've tried tech support at Microsoft, Apple, Adobe or Macromedia, just to be overwhelmed by their cost. And here you are, still puzzled, still stuck, and still having to endure your computing environment rather than enjoy it.

Get roadside assistance with the AnswerSquad!

We're a group of best-selling authors and tech experts who are creating new, innovative and more efficient ways to help you find your computer nirvana: having everything work the way you want, when you want, and how you want.

Visit our website to learn how you can get the answers you need now.

AnswerSquad.com

Graphic icons

Graphic icons: symbols of authority, power and control

Marcus McCallion

There are a lot of important and exciting discussions currently taking place around issues concerning the ownership of ideas. The thoughts and the accompanying practices surrounding the subject have been formed through a diverse range of alliances, interests and motivations. The arguments are becoming increasingly polarised into distinct methods and approaches that already challenge and govern, not only our lives and working practices, but also, our ability to communicate.

It is vital that we as practitioners, within the creative sphere, equip ourselves with the knowledge necessary to understand and actively participate in the debate. To the community in general, whose concerns and discussions may lie within the realm of aesthetic production, these arguments may appear dull, irrelevant, and even trivial. However, if we continue to remain passive and do not seek even a basic insight, we will allow not only ourselves, but also our knowledge and skills to remain as tools for manipulation and control.

Proprietary symbols

A lot has been written about the history of letterforms and their proposed origins. These investigations tend to focus on the technical aspects and the periodical development of typography. There is however very little analysis surrounding the political purpose, ideology or implications of graphic symbols.

Within the present context, the vast majority of alphabetical symbols are intended to operate as devices in aiding the readers understanding. They are a set of tools used to express the authors voice, inflection and tonality, facilitate ideas and give them a means of expression. Despite some interesting and original thinking, there have been very few recent additions to the general lexicon. However, there are some relatively contemporary typographical designs that have blended in so well, we are now unaware of them or their reason for existing.

If we continue to remain passive and do not seek even a basic insight, we will allow not only ourselves, but also our knowledge and skills to remain as tools for manipulation and control

One such design is the copyright symbol - represented by the letter ‘C’ in a circle “©”. With an unimposing and simple design, its place in society can easily be taken for granted. No longer is the primary function of this modern glyph concerned with expanding thought, language, grammar or contextualisation. Indeed, the only meaning this character can convey is that the particular piece of work it is associated with is owned.

Along with other proprietary logos, such as ® and ™, it has been granted a place within the Unicode character set which helps reinforce and maintain a presupposed standard.

In a similar but expanded way, Deleuze/Guattari (1988: 76) stress the disciplinary character of language: "Language is made not to be believed but to be obeyed, and to compel obedience".

What is interesting is, in order for this system to work, the symbol itself, has to be free from these very constraints.

There is no copyright on the copyright symbol. Indeed rather perversely the © device has to be a non-copyrighted design otherwise nobody would be able to use it.

Alternative approaches

The primary concern of copyright laws lies in restricting the right to make copies of a particular work. However, ideas are not objects that can be wholly owned - they can be infinitely copied and re-used due to their immaterial form. That is the essential thought behind the concept of copyleft (all rights reversed). The icon itself is a clever appropriation of the © symbol. By reversing the design, we have to stop and think what the original represents, and what this new form communicates.

Wikipedia explain that "A copyleft license uses copyright law in order to ensure that every person who possesses a work, can also use, modify, and redistribute both the work, and derivative versions of the work. In that sense copyleft is a simple but powerful method for exploiting the established structure of Intellectual Property Rights. By subverting the restrictions of copyright, it imposes on the dissemination and development of humankind's knowledge." - The idea is, in itself, very simple, but its implications are quite dramatic.

Fig. 1: Copyleft is a system that frees information

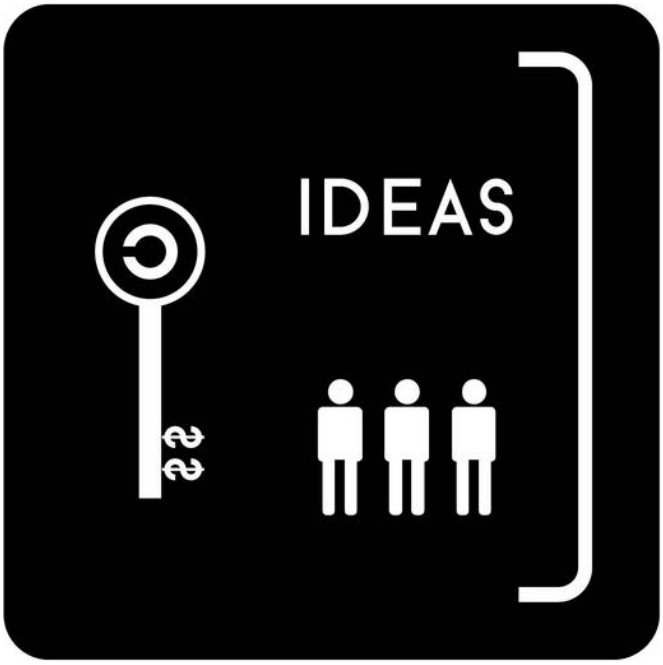
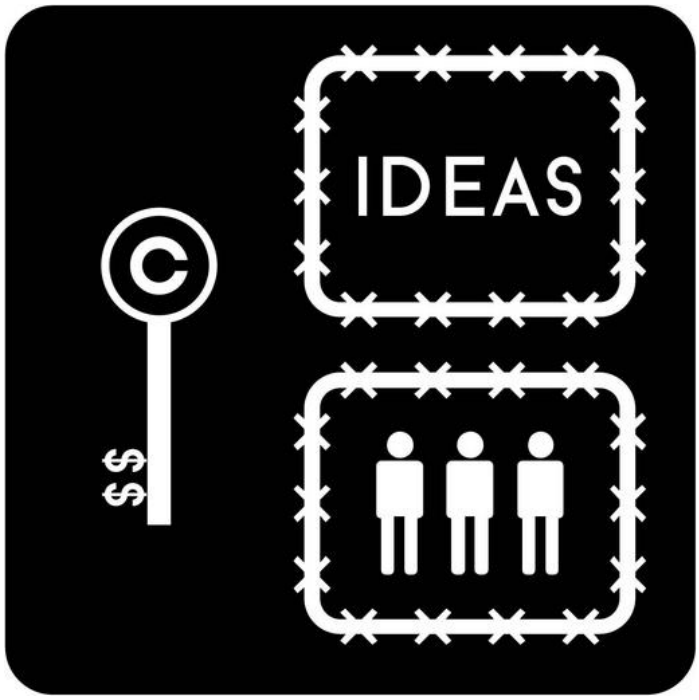


Fig. 2: Copyright is part of a system to control information



Power and authority

Late capitalism is predicated on the creation of scarcity in informational commodities, by the implementation of monopoly, granting intellectual property law, which governs their use. It is highly vulnerable to the critique offered by what seems like a simple solution - information can be infinitely copied and re-used due to its immaterial form. Therefore constricting the flow, copying and creation of informational artefacts is highly difficult and politically challenging for the multinational media corporations.

It is important to understand that releasing work copylefted is not giving things away. Rather, it is licensing informational products, in such a way that their use within commodity capitalism, will result in a transformation of the nature of the derivational goods. In other words, copyleft licensing encourages a chain of goods to become equally open.

This has radical potentials that could produce far-reaching consequences.

Potential

In the present system, there is no single method or approach, which can be initiated to eliminate poverty, slavery or human suffering. This however, should not be used as a reason or excuse to do nothing. If we are the imaginative and creative industry we are so keen to promote ourselves as, we must engage that potential and utilise our collective knowledge and skills, to produce work that has real effect. Explicitly seizing the opportunity to use copyleft licensing is more than just saying you are political. Unlike individualistic and banal statements of intent, this could provide a lever to radicalise and transform our, taken for granted, notions of property.

(With thanks to David M. Berry - Libre Society)

Libre Society's web site (http://libresociety.org)

Copyright information

© 2005 by Marcus McCallion

This article is made available under the "Attribution-Share-alike" Creative Commons License 2.0 available from http://creativecommons.org/licenses/by-sa/2.0/.

About the author

Marcus McCallion is the art director and designer for the independent electronica label LOCA Records. Formed in 1999, this copyleft label has consistently explored, questioned and challenged the purpose of music and art. The innovative and radical design for Wards the single "Armonica or Something", saw the collaborative production of 100 different cover designs. More recently, Marcus has used his knowledge and passion for typography to collaborate with protagonists of free culture - the Libre Society. The result was the production of the first ever copyleft typefaces, called "Posse" and "Profiteer". Acting as differing voices, they were used throughout his "open" design of the Libre Manifesto. Both of these type designs are freely available for download through Marcus's own type foundry ÜNDT

Linux Based Solutions
www.snowdoniait.com
Tel: 01248 355 985 Mob: 07976 632 572
SNOWDONIA
IT SERVICES
By using our services you will gain the many benefits of Linux and free software:
• A secure, stable operating system
• A reduction in licensing costs
• A wide variety of applications for home and office use
• A powerful, customisable system
Services available include:
• Installation and configuration of server and desktop systems with appropriate applications
• Training in the use of your Linux system
• Website design with hosting on Linux servers
• Monitoring systems with customised plugins
• Vulnerability scanning and information security programme development
Contact us for a demonstration of Linux and the applications available, or to enquire about our Linux based solutions for home users and businesses.
On-site daytime and evening visits available. Competitive rates, free estimates.

Free software: working together

Avoiding headaches and lawsuits by working together with free software

Albert Witteveen

Small companies often have to work together; however, nobody wants the headache of contracts, failed promises or deception. Free software allows small companies to work together without these risks, and enables them to create amazing software like WebGUI (http://www.webgui.org/).

How it all began

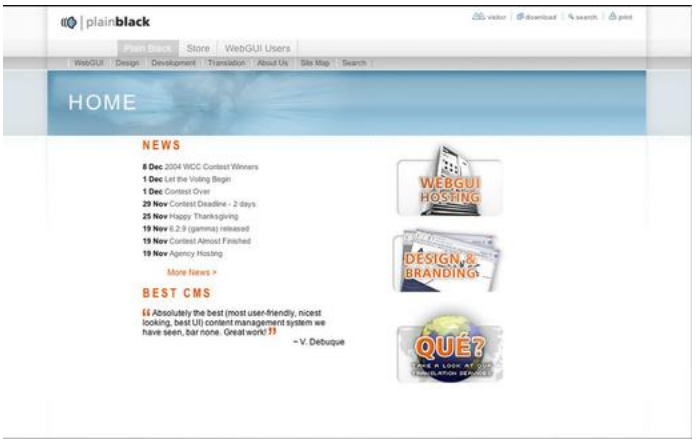
A few years ago we were asked to build an intranet for a client. It was natural for us to use free software and we realised that a Content Management System (CMS) was needed. (We had built a web application before, using PHP, but realised that that language didn't offer what we wanted.) So we looked for an existing CMS that was based on Perl.

We went straight to Freshmeat (http://www.freshmeat.net) where we came across WebGUI (http://www.webgui.org/). Even though it was a relatively new piece of software, it was feature-rich, flexible and a review of the code showed us that its programmers knew what they were doing.

We had been accustomed to building applications from scratch, but after discovering WebGUI we realised that using it to build a site or a web application we save us a lot of time. It offered us many functionalities which we could pass on to our customers at no additional cost.

In the beginning WebGUI was simply something we needed for one small project, but eventually, it came to be one of the most important applications our company offered.

Fig. 1: WebGUI's home page (http://www.webgui.org)



To contribute or not to contribute...

WebGUI wasn't (and isn't) ours; it had been built by an American company. They released WebGUI under the GPL license, but the product's copyright has remained theirs.

Contributing to WebGUI came naturally, but we recognised that building on someone else's product, (albeit open source), could be slightly awkward. It took time deciding to invest in a product that "wasn't ours". Eventually we decided to go ahead, mainly because we felt "safe": If we didn't like the direction taken by the American company for any reason, we could always fork (that is, create a piece of software derived from WebGUI's codebase - the GPL allows it!). We certainly had no intention of forking, but it was

good to know that the option was there. Now we've worked with WebGUI for several years, and we still really don't think we'll ever have to fork!. Several companies and volunteers (including us!) are developing WebGUI at a great pace, and are enjoying the benefits of free software.

Working together

Developing complex software can be a very challenging task - definitely not something you can complete in one lazy Sunday afternoon! There's generally a substantial investment behind developing software - not usually something a small company can do on its own. However, the history of free software has shown that several small companies and individuals working together can achieve the unimaginable If a private programmer (or a company) creates a valid free software project, he or she can get free aid from those who use it, and the results can be astonishing.

The fact that small companies can work together isn't news; what should be known is that free software makes it *easy*. You can start working on a project without the headache of negotiations. All you have to do is write the code, and submit it to the project coordinator. It doesn't really matter if a contributor does less than expected. There is no need to discuss ownership, percentages, etc.

There's also no risk of loosing your investment (in terms of time or money). If any of the companies you are working with go bankrupt or a third party buys one of them out, there are no assets that can be taken away from the project.

There are also some benefits of this process that are normally overlooked For example, free software companies get to know (professionally and personally) the private individuals who contribute to a project.

When these companies need extra staff, they can (and do!) ask the volunteers if they want to turn their hobby into a job. There are several advantages to this: the companies know exactly who they are hiring, how reliable they are, the quality of their work, etc; and these contributors already know the project from the inside!

Another advantage of free software is that having a large number of people downloading and running a piece of software increases its stability substantially - nothing is more effective for finding bugs as feedback from all those end-users. This is one of the reasons why in free software stable releases really are stable.

All of these benefits applied to the WebGUI project - in fact, we discovered them as we went along with the project. WebGUI is a feature-rich CSM - Its development involved many contributors and users, a lot of users' feedback, promotion, etc.

This free software model applied perfectly well to WebGUI: by working together, we now have a product that is saving us millions in terms of development costs.

Together, we can compete with, and outperform, our expensive competitors.

"Use strict"

There are potential stipulations if you are a company and you contribute to a free software project. One of these possible problems is that you won't actually have control over the project. You can always fork, but then you miss out on all the improvements that volunteers and other companies make to the 'original' project.

For example, in the case of the WebGUI project, the code could be described as following the Perl term "Use strict"; There is one person in charge of the project, and he *is* strict: he won't allow any code in that isn't of very, very high quality.

At first, this strictness appeared excessive, even harsh - it may even have scared some contributors away. However, in the long term this turned out to be a blessing. Based on experience, we feel very confident about WebGUI upgrades. Although we still maintain a rigorous "test first" policy for upgrades, no site that uses "vanilla" WebGUI (no custom adaptations) has ever given us any problems.

The fact that small companies can work together isn't news; what should be known is that free software makes it easy

Perhaps, in the beginning, the project might have attracted more developers if we had been a little more flexible. But now it saves the project's developers time and headaches.

For us (as a company), the maintainer's strictness was never a problem. We're glad that no "bad code" finds its way into the project. And if we code something for a client that isn't

Fig. 2: ProcoliX's home page (http://www.procolix.com)

ProcoliX
Creatieve en professionele ICT
WebGUI
Linux
Clusters
Hosting
Software
Beheer
Klanten
Contact
English
Creatieve en professionele ICT
Creatief: Wij denken vanuit het echte probleem van de klant en niet vanuit standaardoplossingen. ProcoliX wil doorgronden wat de klant echt wil bereiken en biedt daar oplossingen voor
Professioneel: Onze medewerkers hebben jarenlange ervaring. Daardoor leveren we niet alleen technisch goede oplossingen, maar zorgen we voor een goede organisatie, planning en documentatie van onze werkzaamheden.
Door de gunstige ligging van Delft ten opzichte van Den Haag en Rotterdam kunnen wij in een groot gebied kwalitatief goede dienstverlening aanbieden.
Onze diensten
WebGUI - Een zeer volwassen Open Source Content Management Systeem en webapplicatie framework.
Linux - Beheer en support van servers en desktops. Wij beheren Linux en bijbehorende Open Source producten, zowel webservers als file- and printer servers en leveren support voor Linux desktops.
Clusters - Clusters en grote systemen: medewerkers met meer dan acht jaar ervaring met het opzetten en beheren van grote systemen bouwen uw cluster of helpen u daarbij. Lees meer over de voordelen-, valkuilen en verschillende soorten clusters.
Hosting - Variërend van ons vlaggenschip, WebGUI hosting tot dedicated servers en onze uitgebreide emailservice.
Software - Software ontwikkelen is meer dan programmeren. We hanteren een duidelijke filosofie die ervoor zorgt dat de klant blijvend waar heeft voor zijn geld en schaalbare en duurzame software krijgt.
Beheer - Linux en Windows systeembeheer en natuurlijk combinaties daarvan. We doen dit met
Kijk in onze shop!
In onze shop hebben we producten zoals:
Linux computers (laptops, desktops en servers)
WebGUI hosting en producten
Linux Training
MySQL ondersteuning
Off-site backup voor MKB
Nieuws
ProcoliX is verhuisd!
- 11/16/2004 3:34 pm -
Door de groei in werknemers en activiteiten werd onze gezellige ruimte aan de Oostplein te krap. Daarom zijn we sinds 15 november verhuisd naar het Radex Innovation Centre in Delft. Dit gebouw ligt aan de rand van de TU-wijk. (Lees meer...)
Lawrence Livermore National Laboratory kiest voor Snapshot EFB
- 10/5/2004 1:16 pm -
Vandaag werd bekend dat Lawrence Livermore National Laboratory kiest voor Snapshot EFB. Het instituut uit Californie is bekend in de supercomputing wereld met meerdere supercomputers en heeft uitgebreid onderzoek gedaan naar welke backupsoftware het meest geschikt is. Een dergelijk instituut stelt zeer hoge eisen aan de backup, omdat men met zeer grote

Textbox 1: Projects going international...

WebGUI became an international product almost instantly, due to the origins of its contributors. In October there was user conference in the USA (the home of its builder). And then, since the product is also supported by us (we're a European company), the next conference will be held in Amsterdam.

considered integral to the project as a whole, then it doesn't make sense for us to keep the responsibility of maintaining the code. We always make code that isn't considered "core" available for download anyway, so the community doesn't miss out on anything.

Conclusion

We at ProcoliX (http://www.ProcoliX.com) ended up building a business around a product we accidentally found on the web.

We don't sell software: we make money providing actual services. Such a model enabled us to start quickly, and with relatively little risk. Thanks to free software: we develop software very rapidly, and deliver full-featured products with prompt and excellent support.

Bibliography

[1] WebGUI: WebGUI (http://www.plainblack.com/wg).

[2] ProcoliX: ProcoliX (http://www.procoliX.com).

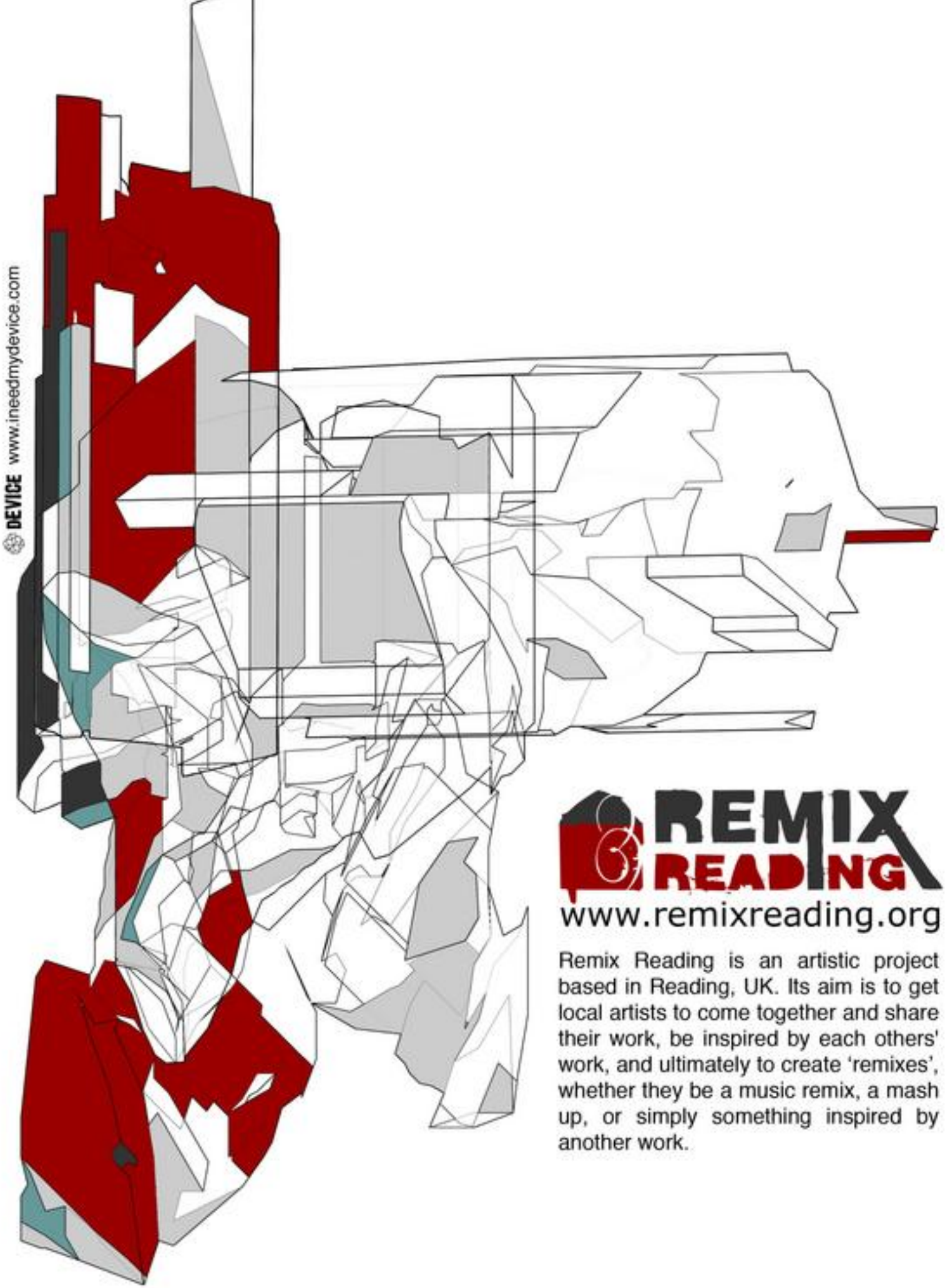
Copyright information

© 2005 by Albert Witteveen

Verbatim copying and distribution of this entire article is permitted in any medium without royalty provided this notice is preserved.

About the author

Albert Witteveen is co-founder and partner in ProcoliX, a growing company in the Netherlands founded in the year 2000 which focussed on free software from day one.



REMIX READING

www.remixreading.org

Remix Reading is an artistic project based in Reading, UK. Its aim is to get local artists to come together and share their work, be inspired by each others' work, and ultimately to create 'remixes', whether they be a music remix, a mash up, or simply something inspired by another work.

The social implications of free software

In the not-so-affluent world in particular, proprietary software deserves to be skipped

Frederick Noronha

If you're new to it, free software appears to be tough to shift to. It also tends to be supported by a smaller pool of techies, and has something of a steep initial learning-curve. So why shift at all? In any case, you can easily make do with illegally-copied proprietary software... right?

Wrong! That's a lazy way of looking at things. It's also an outdated approach, which goes back just three decades or so, when proprietary- you can't copy it, you can't share it - software became the norm.

Today, free software offers so many different alternatives, that it'd be criminal to avoid giving it more attention.

> Proprietary software isn't going to give up without a fight. Their last strategy is to confuse issues. Free software isn't really "free". Why call it free when it isn't always zero-cost? There are a whole lot of similar arguments, which are proliferating in this bitter battle for the hearts and minds of computer users

In a report on free/libre and open source software's impact on the developing world, Finland-based researcher Niranjan Rajani cites a number of reasons why free software makes even more sense to the poorer and resource-hungry countries of the planet. This report, "Free as in Education" Significance of the Free/Libre and Open Source Software for Developing Countries, version 1.0, dated May 2003, is available online on the FLOSS for Development site (http://www.maailma.kaapeli.fi/FLOSS_for_dev.html). The reasons Rajani cites include: lower cost; using free software as a means of coping with the 'anti-piracy' campaign; and deploying it on grounds of security and technological independence.

Tere Vaden, a professor at the University of Tampere's Hypermedialab, argues that the "background motivations" for creating and using free software range from the pure technical (i.e. speed of development, security and privacy, technological independence, ease of use) to the deeply economic, social, political and philosophical (i.e. price, co-operation, equality, commitment to the right to know).

Free software works out reasonably priced both in the short and long term. Free software creates local jobs and multiplies local skills. Free software is transparent enough for you to 1) learn it, if you have the technical background 2) make custom changes in the manner you wish to, or you can pay others to do this for you and 3) enable both you and your staff to learn at a much deeper level.

Real-life examples from South Asia

Looking at the real world easily helps us to understand why free software makes a big difference to its users. Many of the examples below are from the South Asia region. This is due partly to the fact I am based there. Significantly too, the South Asia region is an under-studied re-

Fig. 1: Asia looks at Linux... file photo of an earlier AOSS meeting. Photo by Frederick Noronha

gion in the free software world, and a lot of initiatives from South Asia are simply waiting too long to be studied. An example is Project Ganeshas (http://www.ganeshas-project.org/), named after the elephant-headed Hindu god, realises the potency of GNU/Linux in schools in Nepal and has been working on this front in recent years.

Free software is also helping young South Asians to work collaboratively across continents, forge software and bring out the best in each other. Take a look at an Indian version of Sourceforge (http://sarovar.org/). Likewise, the Project Resource Centre (PRC) on the sarai.net server is working to enhance the skills of local techies.

For a general overview of free software in this region, check out the Free Software Foundation of India (http://fsf.org.in) or Linux India (http://www.linux-india.org). However, more often, the real story lies in the easy-to-miss details.

Using barefoot entrepreneurial skills, college students and other young people are selling copies of free software CDs, at a very low cost - the equivalent of a dollar each and less. In this way, they're earning some money for themselves and doing a favour by spreading the use of free software.

In South India, linuxense.com is a "GNU/Linux-based Enterprise providing software solutions of exceptional quality using cutting-edge technologies; creating a GNU/Linux ambience for our distinguished clients in their demanding work environments."

Free software - sometimes called "open source" by those preferring a more corporate-friendly name - is also contributing to education. "Freed" is a content management system and free education website (http://free-ed.org:12080/Freed/). Educationists are encouraged to create an account for themselves and upload, browse, comment on or rate content available on this site. The objective of this site is to enable everyone to help others and themselves learn.

There are many other ways in which free software enables and empowers. GLUGs (GNU/Linux user groups) build skills among hundreds of young people. One listing in India's "Linux For You" magazine shows that there are around 80 GLUGs scattered across India alone, some more active than others. You can also see a listing of local GLUGs, LUGs, Free Software User Groups (http://wikiwikiweb.de/LugsList) and the like.

Free software is offering other solutions too.

Today, almost all software, operating systems and websites in much of the less-affluent world are in the English and other non-Indian language. In a country like India, since English is a language spoken by less than 10 percent of the population - and is generally considered to be the language of the affluent - language is a significant barrier. This situation is creating a new class of people; those who live in "information poverty" even as technology becomes cheaper and cheaper. To destroy this barrier, techies from the free software camp have been stressing the need to create a national-level, collaborative effort to localise GNU/Linux to Indian languages.

Localisation is taking computing to languages - even small ones - which nobody every dreamt would get access to the power of IT. Networks such as IndicComputing on Sourceforge.net are doing a great job finding local solutions to local requirements, and all without much in the way of financial resources.

So, why is [GNU]Linux in the search for local-language solutions? Simply because its openess scores over the centrally-controlled approached. This allows local linguistic groups to customise user interfaces in ways that are far more culturally sensitive than any centrally-controlled approach. Small linguistic groups, considered too tiny to form a viable market for any vendor, can now also work to tailor

Textbox 1: Why do people develop, code and program free software?

It appears almost unbelievably contrarian and unusual. Who creates free software, and who shares it? Why would anyone create it without the thought of making their million?
Rajani's report says software developers, coders and programmers undertake this task for a range of motives. This includes altruism and a sense of sharing knowledge, taking on a challenging task, doing it just for fun, undertaking the task as something needed to be done for one's own work, for developing new skills, or even in expectation of an indirect reward such as improving job opportunities.

the free software interface to their own needs. The IndicComputing network has argued "We therefore believe that GNU/Linux is a very attractive long-term solution to India's computing needs".
Their logic is self-explanatory. Information technology in India has been confined to the three percent of the Indian population who feel comfortable reading, speaking and writing English. IndicComputing has argued "Given that there are one billion Indian citizens, this is a scandal. [GNU]Linux and its ecology encompass almost all areas of computing today from small embedded devices and clusters with thousands of nodes. We believe that [GNU]Linux technology should be accessible to all Indians, regardless of linguistic background,".
Free software comes with its "four freedoms". This, itself, offers the possibility of unlimited sharing of knowledge. In India, a CD called "Linux In A Teaspoon" - it deals with the entire OS, not just the kernel, though - contains over 40 full-length text books and over 200 "how to" documents covering every aspect of the GNU/Linux in, what its authors call, "a profound but readable style". It also includes over 120 mini- "how tos" and *tons* of tutorial material, plus numerous links and pointers to more resources on the world wide web. As if this were not enough, the CDROM also includes 70 issues of the online *Linux Gazette*.
To put this together, the "Linux In A Teaspoon" team had to compile over 17,000 files in over 3,000 directories. They merged this all in a single, easy-to-use top-level navigational index. Its price - at just INR500 in India (around US$11) - is a modest enough price to basically - say the authors - meet copying, media costs, delivery and direct overheads only. Can you think of any other system of knowledge-building and sharing that would allow for such potent possibilties?

Models for creating, sharing knowledge and solutions

Talent-rich but resource-strapped enterprises, schools and even SMEs (Small and Medium Enterprises) in the less-affluent parts of the globe are finding different models for creating and utilising the potential of computer software.
India is considered to be a superpower-in-the-making in the computer software field. Yet, the high costs of software means that most users in this country simply can't afford or don't use legal products. That's where free software comes in. It's finding a role in powering software development. It can help boost e-commerce, and suddenly make it all the more easy for non-techies to update and maintain their website.
Best of all: this promise comes at a price anyone can afford. Free software doesn't refer to "free beer" - but "free speech". Yet, because it doesn't use the "you can't copy software" paradigm of proprietary software, its price tends to be far more reasonable. Most of the time, it's just a download away from the internet.
However, many who are opting for it, are choosing it not because of its low (or, almost no) costs. GNU/Linux is a very high-quality product, it gives you the chance to get into its innards and work on it. Thus, for many, it is the basic tenets of free software, that make it so attractive. Including the freedom to study, freedom to change, freedom to share or distribute, the right to sell free software, and the principle that the software 'source' has always to accompany binaries.
There are other benefits too.
Dr V. Vinay, of the Bangalore-based Indian Institute of Science, said in an interview some time back, "Children like to play with things, tear them apart and - if we are lucky - put them back together. Free software encourages such exploration, allows interaction with other 'children', and helps them to understand large complex programs. It does this without inducing any guilt of being a 'pirate'! Dexterity in

Fig. 2: Techies throng to GNU/Linux fairs in Bangalore. Photo by Frederick Noronha

creation and not in usage is crucial if a developing country like India has to create its own niche. Or else, we will merely be followers." The other major impact of free software, which Dr Vinay sees, is on the security of the country. "Free software is software that can be trusted as we have the source code," he explains.

GNU/Linux is not important only from the cost point of view. It is powerful and robust. It is also flexible and gives you the power to modify. In addition, Free Software has a very low cost of ownership, contrary to the propaganda and claims put out by the world of proprietary software. You don't pay per client licence, you don't pay for every upgrade, add-on or feature. Once you set it up, it requires very little maintenance. Even if you need to pay someone to customise the software to your needs, the gratification is that this money goes to local talent, not some rich corporation that needs to get richer.

In a region where technology skills are plentiful, but resources are scarce, this makes even more sense.

So far, computers in India have been widening the gap between the haves and the have nots, says Sudhakar "Thaths" Chandrashekharan, an Indian earlier based in Silicon Valley. "Thaths" earlier worked for Netscape. He has been strongly promoting GNU/Linux in India, and is one of the founders of the Linux-India.org network in India. Technologies like these could change things, because it can breathe new life into old hardware and help bridge the gap, he argues.

In the non-profit world

In the world of volunteering, or among NGOs (Non-Governmental Organisations, as they're called in some parts of the globe), free software has a special role to play.

Free software goes well with the NGO-oriented principles of sharing freely, re-use and waste minimisation. So, is there any reason for NGOs to take the side of a global monopoly? Taking this cue, an international camp for NGOs (http://www.tacticaltech.org/asiasource) interested in free software was held in end-January 2005 in Bangalore, South India.

"Freedom" is something NGOs always talk about, in whatever form. But in the software world, this is already a reality. The possibility exists: are we ready to take a little extra trouble (the initial learning curve) in opting for it? Like its other users, NGOs also get the promise of quality, stable software from the free software world. NGOs too tend to be rich on talent, and limited on (financial) resources. So it makes sense for them to give free software a serious look.

Importantly, free software also empowers computer users and encourages them to cooperate, as Richard M Stallman points out.

Above all, Free Software is an ethical choice - not one of convenience. NGOs also tend to receive, store and disseminate huge amounts of information. It helps to be able to access info (in digital format) without having to 1) break the law, and 2) spend money to purchase applications to "read" the information. Using free software enables that, as South India-based lawyer Mahesh Pai points out.

Groups like Oneworld.net, a coalition of developmental organisations with a regional headquarters in New Delhi, are open to convincing about the need to spread the ideas of free software among NGOs in India (personal email, January 2003).

In December 2002, Steven Sy wrote (in personal email) about Greenpeace-Asia's shift over to free software. Greenpeace Southeast Asia - GPSEA, was formally established on March 1, 2000 and has offices in Bangkok, Thailand (head office) and in Manila, Philippines. It currently has 18 staff in Bangkok and eight in Manila. Globally, Greenpeace runs "ninety percent plus" of their servers on GNU/Linux. But, Manila became the first Greenpeace office to fully deploy GNU/Linux as the majority desktop "The office has been planning to move to free software since early 2002. We

made a conscious choice between migrating to free software or spending funds on expensive software licenses. We also did not want to get into legal troubles if ever the BSA (Business Software Alliance, the proprietary software arm that fights illegal copying of software, which it terms 'piracy') came our way," explains Sy (27). Free software was used for word processing, e-mail, web browsing, spreadsheets and presentations.

The advantages are obvious: "It's free (both as in 'beer' and as in 'speech') and secure (less or no virus infections since migrating)." The motives for shifting over: "It's a mix of both (technical and philosophical). Free software is a technically superior and morally correct technology," says Sy. All of the regular staff now use free software in their daily work.

"Since we downloaded the software off the internet, just the costs of blank CDs that's less than US$1. For the users, just time and patience in learning the new system," says he. "Free software saved the office a lot of money, money that was better spent on winning campaigns (for the environment) than paying for very expensive licenses." Problems have been limited to "some minor bugs in the software and steep learning curve for administering (for a beginner)."

Conclusion: opening the doors

Prakash Advani, a long-term supporter of Free/Libre and Open Source Software, and earlier head of the Freeos.com network, has argued that GNU/Linux is being increasingly accepted by the corporate world, both as a tool to save money and offer cutting-edge technologies. Advani said that with 50 users, any firm could save considerable sums by using a Linux print-and-file server, as an internet and e-mail server, and also for networking. "It can also be used for web-servers, proxy-servers, internet servers, firewalls, routers, application servers, database servers or fax gateways," he has said.

Syed Khader, who has worked with IBM-Bangalore, tells how GNU/Linux had been put on a wrist-watch at the IBM's Bangalore research centre. In past exhibitions, he has demonstrated super-computing with GNU/Linux. Khader showed how computers could be linked in clusters. Such tools have immense applications in fields like weather forecasting, unearthing the genome code, or coping with powerful web-servers, he notes. GNU/Linux could thus be used to support web-server processors that could take upto one million hits per second.

"[GNU]Linux reduces hardware costs, and requires less maintenance. It involves no cost for upgrades. There are also savings on add-ons like anti-virii programmes," says FreeOS.com CEO Prakash Advani. As more of the planet wake up to its potential, there's a growing realisation, that in a world dominated by Windows, that GNU/Linux is indeed opening many doors.

Can we dream of a time where most of the software used by the average user is free?

Fight as the proprietary lobby might - using all kinds of tools, from software patents, to "trusted computing" - such a scenario is not wholly unlikely or impractical. As computing stretches to make its impact felt on the lives of a wider section of the planet, there's little choice but to give free software a good, hard look.

Some could see free software as an affordable solution, with no restrictive licensing barriers. Others will look to it for the promise of stability and technological efficiency. Techies will, no doubt, continue to see it as a fun way of learning, sharing knowledge, and contributing their skills to those who need it most.

Put together, what could this mean? Less of a "digital divide". Fewer computers consigned to mountains of e-wastes, just because the planned obsolescence needs of proprietary software makes them useless long before their time. Easier access to youngsters to learn the innards of computing, not just some superficialities. Smaller languages of the planet suddenly finding computing relevant to them... and more.

What would life be without its dreams and hopes?

Copyright information

© 2004 by Frederick Noronha

Verbatim copying and distribution of this entire article is permitted in any medium without royalty provided this notice is preserved.

About the author

Frederick Noronha is a freelance journalist based in Goa, India. He regularly writes articles on free software. He is a co-founder of Bytesforall (www.bytesforall.org)

qreate

Graphic, Web, Life design

PRINT

Free Software
MAGAZINE
CREATING FREE SOFTWARE MAGAZINE
A COMPLETE AUTOMATED SYSTEM TO CREATE MAGAZINES BASED ON FREE SOFTWARE
INTERVIEW WITH MYTHIC BEASTS' STAFF
THE SECOND MAC REVOLUTION
THE CONTENT TAIL WAGS THE IT DOG

FREE SOFTWARE MAGAZINE'S
ISSUE 0 COVER

FREE SOFTWARE MAGAZINE'S
MEDIA KIT

FREE SOFTWARE MAGAZINE'S
ADVERTISEMENT

WEB

COLLE MORO BED&BREAKFAST
www.collemoro.com

VILLA DELLE RONDINI
www.villadellerondini.it

www.qreate.net ✵ *info@qreate.net*

Free Software MAGAZINE

BY SUBSCRIBING YOU WILL BE SUPPORTING A MAGAZINE WHICH BELIEVES IN FREE SOFTWARE. ALL OUR ARTICLES ARE RELEASED UNDER THE GNU FREE DOCUMENTATION LICENSE, ENHANCING EXISTING INFORMATION ON FREE SOFTWARE.

SUBSCRIBE NOW!

HTTP://WWW.FREESOFTWAREMAGAZINE.COM/SUBSCRIBE